THE BEST SELLING COMPUTER PROJECTS MAGAZINE

Big Brother is here!

JANUARY 1984

ELECTRONICS & COMPUTING MONTHLY

AN EMAP PUBLICATION

10
12 NEWBRAIN

Germany D5.80
U.S.A. $2.95
85p

5 SPECTRUM μ SPEECHES TO BE WON
FREE TO ENTER

IS BIG BROTHER DIGITISING YOU?

9 FREE ADVERTISNG
10 COMPUTER COLLAPSE
12. BARCODE REF
13 PRINTER, RTTY, ROM & MORSE DECODER
18 VOICE SYNTH.
23 ORIC HIRES
30 ELECTRON I/O PORT
34 REF
37 SAT 16 COMPUTER SYSTEM
42 MICRO GRAPHICS
46 RAM BOARD
53 CROSS ASS. 68705
57 PROBE KIT
58 VIDEO CAMERA INTERFACE
60 SAT-16 SYSTEM

64 PRINTER
71 REF
78 KIT & ROBOT BOARD
79 EPROM REF
80 Z80 / 6502 VECTORS
88 SANYO RECORDER
89 FREE ADVERTISING

BBC VIDEO INTERFACE REVIEW

PLUS: BBC SIDEWAYS RAM SOFTWARE

BUILD AN ELECTRON I/O PORT

SPECTRUM SPEECH PROJECT • 68705 CROSS ASSEMBLER DESCRIBED

SCIENTIFIC EXPERIMENTS WITH THE BBC • ORIC HI-RES GRAPHICS

Electronics & Computing Monthly
Scriptor Court,
155 Farringdon Road,
London,
EC1R 3AD

EDITORIAL
01-833-0846
Editor
Gary Evans

Assistant Editor
William Owen

Administration
Liz Gregory

ADVERTISING
01-833-0531
Manager
Claire Fullerton

Executive
Richard Jansz

Classified
Andrew King

PRODUCTION
01-833-0531
Art Editor
Jeremy Webb

Make-up
Time Graphics

Publisher
Alfred Rolington

Distribution
EMAP
National Publications

Published By
EMAP Business And
Computer Publications

Printed by
Riverside Press,
England

Subscriptions
Electronics &
Computing Monthly,
(Subscription
Department),
Competition House,
Farndon Road,
Market Harborough,
Leicestershire.

ABC
MEMBER OF THE AUDIT
BUREAU OF CIRCULATION

Vol. 4

IN THIS ISSUE

Issue 1

Page 75 has all the details of our latest competition and page 89 carries details of a brand new *E&CM* service.

We apologise for the fact that Part Two of the Microtan 65 has been held over due to lack of space.

Coming Soon . . . Your Robot . . . . See Page 28

Electronics & Computing Monthly is normally published on the 13th day of each month

EDITORIAL

Despite the fact that this issue bears the date January 1984 it is scheduled to appear during the dying days of 1983. It therefore seems appropriate for me to take this opportunity of wishing all our readers a Happy Christmas and good wishes for '84 on behalf of all the staff at *E&CM*. I shall not however indulge in an assessment of this past year, nor look forward to the next as Mike James, a familiar name to regular readers, covers this ground in his article 'Into 1984' that starts on page 68.

Your Robot

Electronics & Computing Monthly has always prided itself on its innovative and lively editorial coverage. We like to think that we lead and others follow. We continue this policy next month by including a special supplement dealing with all aspects of the field of robotics. In fact supplement is not the right word as 'Your Robot' is the first issue of a new magazine that will appear quarterly and be given away free with *E&CM*.

There can be no doubt that during the next few years there will be a tremendous growth in the field of robotics and related areas. At present most low cost systems are little more than expensive executive toys but this situation shows every sign of changing very rapidly.

If you want to keep in touch with this rapidly expanding area, make sure you read 'Your Robot' next month.

Readers Exchange

A letter from Mr. John de Rivaz of Cornwall makes the point that many individuals with items for sale find the cost of advertising in magazines an uneconomic proposition because of the low cost and limited quantities of the items they wish to dispose of. We take the point to heart and will shortly introduce a new service whereby readers can advertise in the pages of *E&CM* completely free of charge.

Full details of this service appear elsewhere in this issue and we very much hope that readers will use the facility to buy and sell a wide range of computer hardware and software.

The adverts will appear on a first come, first served basis so those wishing to use 'Readers' Exchange' should put pen to paper right away.

Spectrum RGB Converter

We had planned to publish a design to convert the UV and Y video signals of the Spectrum into standard RGB signals in our October issue. Unfortunately the original design was not up to the standard we had hoped and the decision not to go ahead with the project was taken.

We are however working on a new approach that looks promising and we hope to be able to publish this revised design early in 1984.

Gary Evans

5th generation language on the Spectrum

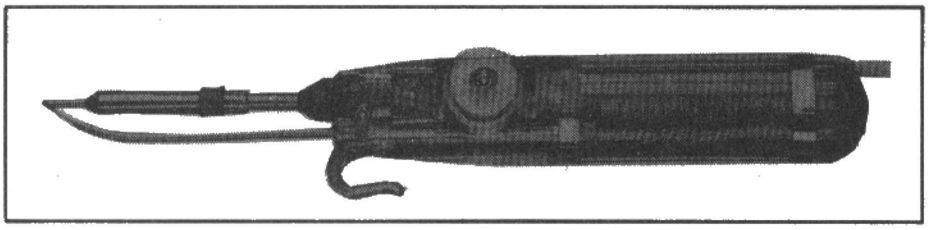
An adaptation of PROLOG, the 5th generation language developed for use in artificial intelligence and knowledge based systems, is now available on the Sinclair Spectrum.

Micro-PROLOG was developed for Sinclair Research by Logic Programming Associates. The difference between this version and other PROLOGs is essentially one of syntax and the allowed form of query. PROLOG is a high level language based on logical data description and data query (PROLOG stands for PROgramming in LOGic). The language employs 'English' phrases rather than sequences of instructions, making possible a relatively direct dialogue between user and computer using familiar concepts and ideas. The information used in Micro-PROLOG applications is stored in a database, which can be extended to answer increasingly complex questions as the user becomes more advanced.

One thought that immediately occurs, however, is that the operation of such a high level language on a Z80 based 48K machine will be extremely slow. The next issue of *E&CM* will carry a review of the package and its usefulness will be fully assessed then.

Micro-PROLOG is available in cassette form, and comes with a user manual and a copy of a 300 page 'Micro-PROLOG primer'; the complete set costs £24.95 and is available by mail order only.

Sinclair Research Ltd., Stanhope road, Camberley, Surrey GU15 3BR.



One handed soldering?

This new soldering iron design will be a boon to novices in electronics construction as well as those clumsy old hands. The GPE self-feed soldering iron enables the constructor to solder with one hand. Flux cored solder is automatically fed, by a spring mechanism, through a stainless steel tube to the bit. The solder is stored in the handle of the iron, and the supply is easily replenished by removing the end of the handle and inserting a refil. A clean bright joint is guaranteed every time, say the manufacturers.

Gardner Precision Engineering, Horht Road, Woking, Surrey.

Spectrum speech recognition

Orion Data Ltd. have made what could become a significant development in home computing. You can say goodbye to those tiresome joysticks and keyboards and tell your computer what to do.

The company has manufactured a speech recognition system called 'Micro-Command' which will retail for under £50. Micro-Command translates commands spoken via a microphone into signals 'instantly' obeyed by the computer. To accommodate different accents, pitch and pronunciation a voice print is taken of each new operator. The commands can be given in any language – opening up the possibility of overseas sales.

The device is currently available only for the Spectrum, but manufacture is currently underway of units which will be compatible with other machines. The potential of Micro-Command will doubtless first be realised in computer games, replacing manual keyboard and joystick control. Instead of pressing a button the games player will be able to yell 'Fire' into the machine: a sobering thought, but this is the type of development which will find many varied applications of its own, both practical and obscure.

The all-inclusive package consists of a speech recognition unit, instruction leaflet, microphone, and cassette (with speech loading instructions and a game).

Orion Data Ltd., 3 Cavendish Street, Brighton, E. Sussex.

Electronics & Computing Camps

A new venture in the area of electronics and computing is starting this summer at Letton Hall – an 18th century country house in the Norfolk countryside. Paul Beverley, who will be known to readers for his articles on the BBC micro, is running a holiday for 13-16 year old children which aims to give them the opportunity to learn more about the link between computers and electronics.

The one week camps each cost £48; dates are 11th – 18th and 18th – 25th August. For more details, contact: *Mrs. Sue Beverley, 57 Cambridge Street, Norwich, NR2 2BA.*

Jupiter collapses . . .

The latest business casualty of the home computer market is Jupiter Cantab, the company established to market the Forth running micro, the Jupiter Ace.

The Ace was launched in late 1982, and has had a life span of less than a year. The computer was greeted with some enthusiasm at its launch from sections of the press as well as FORTH afficionados, but this has not proved enough. According to the manufacturers, who went into liquidation in mid-November, problems arose when the company over-expanded into UK and European markets; large orders were placed but either not taken up or left unpaid. The consequent strain on company finances made further research and development impossible, and the possibility of moving into the control and robotics markets to which this computer was ideally suited, were lost.

TI99 bites the dust . . .

Big companies too are not immune from the vagaries of the home computer market. Hot on the heels of Jupiter came the announcement that Texas Instruments were making their escape with the withdrawal of the TI99/4A. This ROM cartridge based machine has lost millions for TI, and dealers in the UK are currently selling it at rock bottom prices to offload stocks during the Christmas rush.

The TI99/4A might therefore become a good buy, excepting that it will lack software and peripheral support in the future. For current owners, TI's European Consumer Division marketing manager makes the assertion that, 'regarding software, we have sufficient stocks to meet market demand in the foreseeable future both for existing titles and many of those recently announced'. Supplies of TI peripherals are limited but numerous third party companies supply comparable/compatible products.

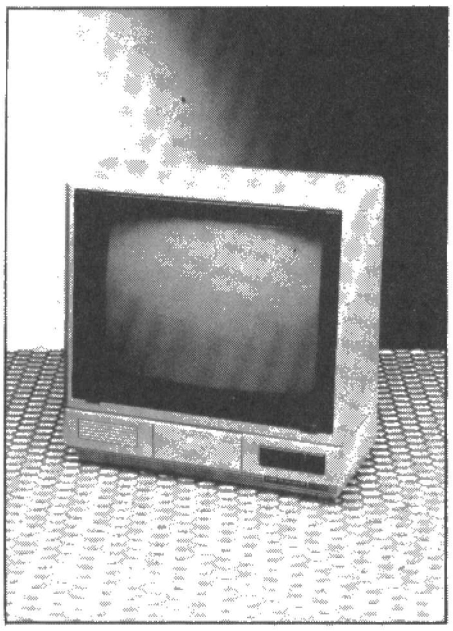
. . . and Newbrain renewed

Newbrain owners who were left out in the cold when Grundy Business Systems collapsed earlier this year can breath again. All design and marketing rights to the Newbrain computer have been bought by a Dutch company, Tradecom International B.V. Tradecom say they are confident that future sales of the machine and its peripherals will ensure continued manufacture and technical development. The revitalised (financially, not technically) machine will be in the shops in the near future.

Colour Monitor

Micro Peripherals 14 inch colour monitor has a 'super bright' screen and allows RGB, RGBY, PAL Composite and audio signal input, providing a 2W output.

The CM14 has a price of £199; its range of input facilities make it useable with a wide range of micros including



Apple, IBM, Sirius, BBC, Dragon, Oric, Lynx and Electron as well as home video recorders.

Micro Peripherals Ltd., 69 The Street, Basing, Basingstoke, Hants.

Newbrain Modem Interface

A characteristic of the Newbrain is that it's video output turns off while accessing its comms port. This renders it next to useless for acting as a terminal in any application. The solution to this is an independent RS232C port on the main expansion bus. This leads on to the question of how to drive it. A package called "Comm Software" has been developed to enable the new RS232C port to couple up to a Modem or Accoustic Coupler and access any bureau computer or electronic mail service.

To date this combination has been used on Telecom Gold, Comet and other 300 baud services over BT telephone lines. Work is taking place on providing the viewdata facility on Prestel. The cost of the Modem interface is £69.50 plus VAT. External power supply for the interface is £6.95 plus VAT.

Kuma Computers Ltd., 11 York Road, Maidenhead, Berks, SL6 1SQ.

ZX Spectrum extensions

A number of new extensions for the ZX Spectrum are available from EPROM Services. The ZX Rampack Adaptor fits to the rear port of the ZX Spectrum and produces suitable decoding to fit most memory mapped ZX81 devices, and the ZX81 RAM pack to the ZX Spectrum. These devices are now relocated to the 32-48 area of memory.

The ZX Extension board is a 3 slot mother board, containing voltage regulation, and power-on indication. It fits to the rear port of the Spectrum, and can accommodate three peripheral cards plus ZX Printer.

The ZX Spectrum EPROM board can accommodate up to 8K of 2K EPROM or 2K RAM ICs (2716 EPROM/6116 RAM). The memory is located at 56-64K but a simple wire-link modification can be made to relocate the board to the 48K area. Thus two boards in series can be made to fill the 48K-64K area. This board plugs straight into the Spectrum and does not require the Extension board. The 48K Spectrum requires a minor internal modification in order to provide a RAM deselect line.

The Auto Start card gives an automatic jump to location FD00 hex on switch-on. The Spectrum initialises prior to the jump, and after executing the machine code held at location FD00, can return to BASIC.

EPROM Services, 3 Wedgewood Drive, Leeds, LS8 1EF.

Interface Kit

Hobbyboard, the hobby electronics division of Kelan Electronics, have launched an interesting interface project kit for the ZX Spectrum. The kit consists of a black box in a shape Spectrum users will recognise as typical of ZX add-ons. Inside the box is a prototype board, edge connector and extender PCB, and a 9-pin D socket (Atari type). The kit could be used to build a joystick controller, RS232 interfacing, relay controller, A-D converter etc. etc. The kit is priced at about £9.00.

Kelan (Hobbyboard), North Works, Hookstone Park, Harrogate, North Yorks HG2 7BU.

Barcode programming

The US company Databar Corporation has announced a potentially important new concept for programming home computers. The system consists of a compact, low-cost optical scanner (called "OSCAR" for Optical Scanning Reader) along with a comprehensive program of software support.

One of OSCAR'S design objectives is communication with most of the popular home computers such as Atari, Commodore, TRS 80, TI 99/4A, Timex-Sinclair, etc. (as you can tell from the list Oscar is geared up for the US market). OSCARS' BASIC software will be compatible with these machines, and it will be uniquely formatted in bar code to permit the average home computer user to enter entire programs in times typically less than two minutes.

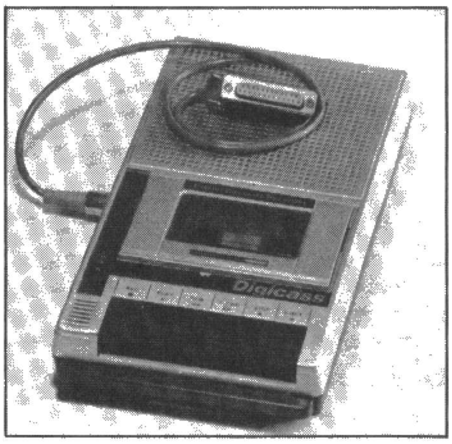
Suggested retail price for OSCAR is $79.95 (£54.00) which includes the premier issue of a new monthly magazine that will contain BASIC programs which can be scanned by OSCAR and transferred to computer memory, eliminating keyboard entry. One particularly useful feature of OSCAR is that it connects to the computer's cassette port: there is no RS232 requirement.

Databar Corporation, 10202 Crosstown Circle, Eden Prairie, MN 55344.

Digital recorder

A new version of the Digicass family of digital tape recorders, the Digicass 1 has an improved performance and is being offered at the price of £249.

The recorder features Phillips-type C60 audio grade cassettes as the recording medium, capable of storing up to 400,000 characters at 1200 bps. Tape speed is 47.6mm (1⅞ inches) per second, offering between 110 to 1200 bps (bauds) and higher data transfer rates. Cassettes can be interchanged freely among any of the



Digicass family. The Digicass 1 is a non-intelligent digital tape recorder requiring all formatting to be implemented externally. The format and data rate employed in recording will be faithfully reproduced on subsequent playback.

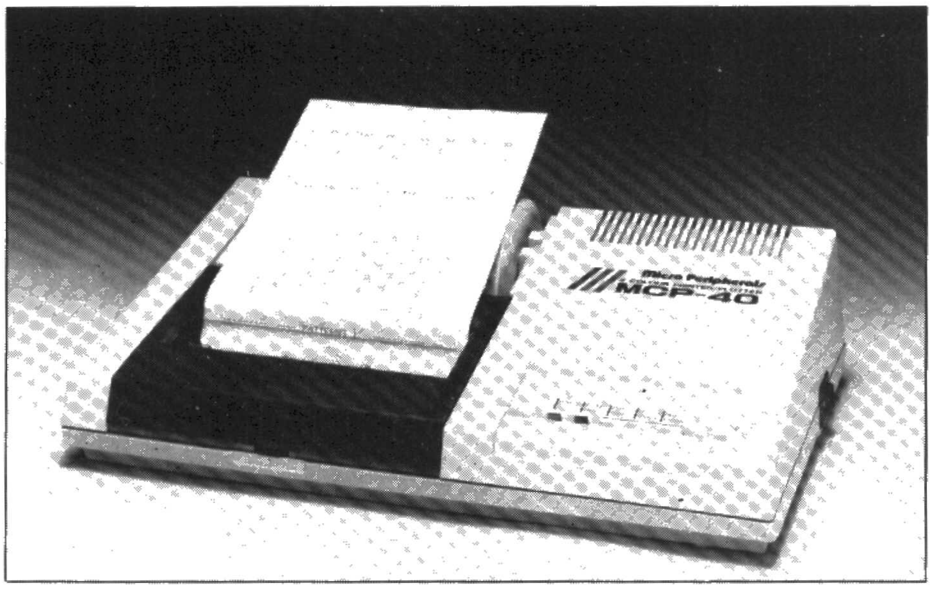
P.A.L., 19 Bayliss Close, Northfield, Birmingham B31 2XP.

Fast Disc Drive Unit

Byte Drive 500 is intended as a new solution to the problem of providing home and personal computer owners with a fast – yet easily controlled – Floppy Disc Drive.

The F.D.D. Units used in the Byte Drive 500 system are single sided, double density 3" CF Compact Floppy Discs, with a formatted capacity of 220K Bytes per side in 11 sectors. This gives a total of 440K Bytes. Typical access time to files is 3ms. The recommended retail price is £260 + VAT.

ITL Kathmill Ltd., The Old Court house, New Road, Chatham, Kent ME4 4QJ.



4 Colour printer/plotter

Micro Peripherals have launched a new 4 colour printer/plotter called the MCP-40 which will retail for £113 ex. VAT.

Charts, graphs, and 4 colour printing can easily be performed by interfacing the lightweight MCP-40 to any parallel output microcomputer.

Up to 80 characters per line at a speed of 12 characters per second, can be readily achieved.

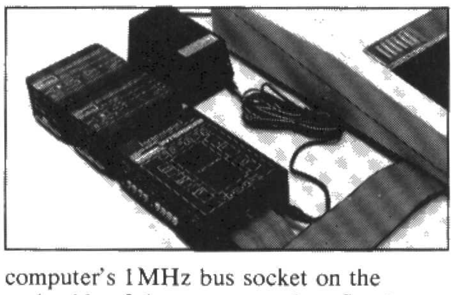
The MCP-40 is suitable for connection to the BBC, Electron, Dragon, Oric, Sinclair Spectrum and Apple.

Micro Peripherals Ltd., 69 The Street, Basing, Basingstoke, Hants.

Interbeeb

The Interbeeb set is designed for use with the BBC Micro Model B and provides a complete interface package for the computer.

The main pack in the set plugs into the



computer's 1MHz bus socket on the underside of the computer via a fitted ribbon cable assembly and polarised 34 way transition connection. It is housed in a neat compact plastic case and contains an 8 channel 8 bit analogue to digital converter, 8 bit input port, 8 bit output port, 4 switch inputs, 4 relay outputs and DCP bus connector for further expansion. A separate DCP Power Pack is also provided in the set which plugs into the mains supply to provide safe low voltage D.C. for the main Interbeeb pack.

DCP Microdevelopments Limited, 2 Station Close, Lingwood, Norwich NR13 4AX.

Display Controller

A display controller for the EPSON HX-20 portable computer, the HO-20, is now available from Oval Automation Limited.

The HO-20 enables the HX-20 to be interfaced to a television or video monitor, permitting the display of 512 alphanumeric characters or over 8,000 colour pixels. It is the only colour graphic display controller approved by EPSON for use with the HX-20.

In the alphanumeric mode of operation, the HO-20 displays 16 lines of 32 characters, in either green or orange.

In the colour graphics mode, the HX-20 displays 128 by 64 colour pixels, each of which can be any of 4 colours, with 2 colour sets available. The 2 sets cannot be intermixed on the display. The background and foreground colours can be independently selected from within the current set.

Oval Automation Limited, Courtwick Lane, Littlehampton, Sussex BN17 7PA, England.

RTTY unit

There is now a great movement in ham radio circles towards the reception of radioteletype (RTTY) using computers (see the soft products section for example). This allows both ham radio transmissions as well as the commercials like TASS or REUTER to be copied in plain language on to a computer's monitor. A new unit has been dedicated to the BBC which has a high level of ownership in ham radio and short wave circles.

The unit, from G3LIV, has duel shift and is able to copy signals that are transmitted in the wrong sense at the transmit end. It is packaged in a very smart container only 150 x 90 x 33mm. Transmit circuitry is also catered for within the unit for the licensed ham.

J. Melvin, G3LIV, 2 Salters Court, Gosforth, Newcastle, Tyne & Wear, NE3 5BH.

SOFT PRODUCTS

BASIC learning package

A cassette and manual based learning package by Logic 3 is now available for Spectrum and Dragon owners, and an Electron package is to be launched in the near future. 'Learn BASIC programming on the Sinclair ZX Spectrum' includes a 200 page A4 manual and two cassettes (20 programs in all). 'Learn BASIC' was written under the supervision of Professor Andrew Colin, originator of the Strathclyde method of programming tuition.

The complete package is priced at £12.95.

Logic 3, Mountbatten House, Victoria Street, Windsor SL4 1HE.

Beebfont ROM

Strictly speaking, Watford Electronics' Beebfont is not soft but firmware. The Beebfont ROM, a 'new concept' according to the manufacturers, gives 5 16x16 pre-defined fonts. It is said to be ideal for high quality demonstration on screen and when used in conjuction with an EPSON printer. Among the fonts available are Gothic, Old English; bold, light and outline sans serifs, and a form of Times Roman. Included within the character sets are accented letters. The package includes an editor on which the user can design his/her own fonts, and several spare fonts which could not be fitted on the ROM but can still run from RAM. It is supplied complete with ROM, software on disc or tape, and a manual.

Watford Electroncs, 33/35 Cardiff Road, Watford WD1 8ED.

Morse Decoder

The G4BMK Morse transceive program (Dragon and TRS80), when used with a suitable interface will decode Morse signals from a radio, displaying the resultant text on a TV or monitor screen. Full transmit facilities are included, and the program will convert text typed on the keyboard into Morse, sending at speeds of up to 200 words per minute (and therefore is suitable for 'meteor scatter' operation).

This and a number of other programs for radio amateurs have been developed by M. Kerry. Also available are a Morse tutor program (for the Dragon) and RTTY program (Dragon and TRS80) from:

M. Kerry G4BMK, 22 Grosvenor Road, Seaford, E. Sussex BN25 2BS.

3D Invaders

Three Dimensional games are now coming into their own, and one of the first for the Oric is 3D Invaders from Quark. According to the publishers the program was developed using a 'specially developed machine code technique' – an intriguing statement – and the 3D achieved by using a novel 'hidden-line' technique. Eight different Invader screens are incorporated into the game.

IBM PC Junior preview

IBM has thrust itself into home computing with the launch of the PC Junior. Can this 16-bit, 128K machine challenge Acorn and Sinclair's domination of the British market?

The IBM Personal Computer has taken the 16-bit business market by storm, with perhaps the one solitary exception of Britain where a strong home market exists, with machines such as the Sirius and now the ACT Apricot holding their own against all the odds. This situation is almost certain to be repeated by IBM's first strike attack into the home computer market – in the form of the Peanut, or as it is now known, the PC Junior, launched in the US in November.

The PC Junior is a new departure in the home computer world: it is expensive, fast, and powerful: the first of a new generation? Two models are already available in the States, but they will not be in the shops in this country until March or April at the earliest. The 64K RAM cassette and cartridge based entry model is priced at $669 (£450) and the 128K RAM model, complete with single disk drive (but with a number of essential extras absent) is priced at $1269 (£860).

The 16-bit home micro

The central processor is the same as that used in the IBM PC: the 16-bit Intel 8088, which runs at a clock speed of 4.77MHz and 210 nanosecond cycle time.

The 64K ROM contains Microsoft BASIC, diagnostic test routines and some elements of PC-DOS. PC Junior is well endowed with interfaces and expansion potential. There are two ROM cartridge slots, and sockets for cassette recorder, audio amplifier, light pen, TV lead, composite video monitor and joysticks. There is an RS232 interface, RAM expansion socket, and two other system board sockets for disk drive controller and modem (the IBM modem is of a high standard and a steal at $200/£135). There is also a full expansion bus offering unlimited possibilities.

"the keyboard has totally independent cordless operation"

One apparently disappointing aspect of the PC Junior is its keyboard. This contains 62 keys (all the IBM PC keys minus function keys) but these are squashed into a small board and their operation does not exactly resemble that of a typewriter – the keys are rubber-topped and have no positive feel.

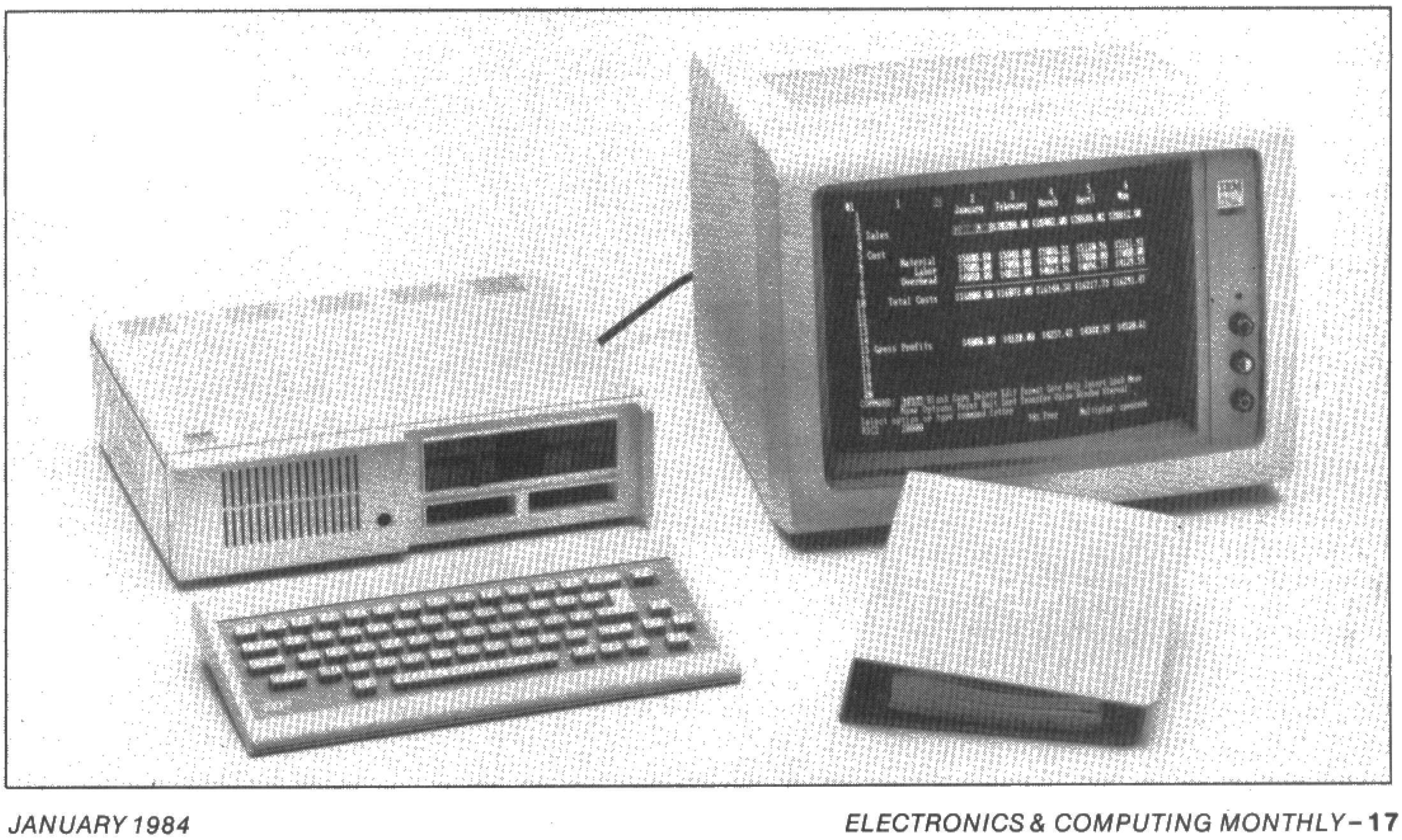
A unique, if gimmicky feature of the keyboard is that it has totally independent, cordless operation. It is connected to the main processor via an infra-red transceiver, with a range of some 20 feet, and is powered by batteries rather than from the main power supply.

To take full advantage of the colour graphics and sound features it is necessary to buy extra cartridges. With these, five different graphics modes are available, offering, for example, hi-res 4 colour or medium-res 16 colour. Normal text display is 40 columns wide.

The IBM PC sells in vast quantities on the basis of the name behind it and the mammoth library of software available. Junior has the name, but only some of the software. Currently there are just four cartridge based games available. However the Expanded Model may run most IBM PC software, subject to certain restrictions caused by, for example the lack of function keys. At Junior's launch 34 compatible PC programs were named.

Who's afraid of the big bad Peanut?

That then, is a brief description of IBM's version of a home computer. Are Sinclair and Acorn quaking in their boots ready to be crushed by the big bad wolf of the computer business? The answer I am sure is no. Both companies can be expected to bring out comparable machines before 1984 is out, and are quite able to meet the challenge. At the moment Junior is over priced for the very strong British market – (and the UK price is likely to be significantly higher than the current US price). It is more likely that it is Acorn and Commodore who are busily battening down the hatches, and they will fare far worse in the States where IBM can, without doubt, expect to dominate home computing in the same way that they dominate the mainframe and personal computer business.



Spectrum Voice Synthesiser

Until recently a voice synthesiser attachment to a microcomputer would have cost at least £200. With the introduction of the General Instrument SPO256 chip a good synthesiser can be built for as little as £15. In this article Robert Harvey outlines the construction and use of just such a device for the Sinclair Spectrum.

The system to be described is intended for connection to a ZX SPECTRUM but the circuit can easily be interfaced to any Z80 based micro and as the driver software is written in BASIC, this too should be transportable with little modification.

The circuit can be easily constructed on VERO VQ board using a wiring pen (also from VERO) or a PCB can be used. The dual 28 way edge connector can be soldered directly to the board and thus the finished unit (with the speaker mounted on the board) can plug in and stand up behind the Spectrum. The circuit includes a five volt power supply for the logic and it is recommended that this be used because the SP0256 chip alone uses around 90mA and it is well known that the Spectrum's own internal power supply will not stand much more loading!

The main circuit consists of three parts: Computer interface, Voice synthesiser chip and the filter amplifier circuit.

Interface

The synthesiser is interfaced to work on one of the Z80 I/O ports: When writing to this port, data is transferred to the SP0256 which then begins pronunciation. When reading the port the Busy (ie. Currently talking) signal is presented on bit 7. The actual address of the port can be changed to any of seven different addresses so that the circuit can be accommodated with any other devices connected to the computer. The three address bits A5, A6 & A7 not used within the Spectrum are used to select this port address. The software assumes port 159 is used.

The Synthesiser

This section of the circuit consists of the SP0256 chip and an oscillator made from two gates of a TTL quad NOR gate IC. GI recommend that a 3.12 MHz crystal be used and provision has been made within the chip to connect this directly without the need for external circuitry. Although, by using a separate oscillator (eliminates the need for an expensive crystal) the pitch of the synthesised voice can be changed to give the most pleasing results. The crystal, if required, can be connected between pins 27 and 28 of the SP0256. The chip itself contains all the logic to convert the allophone codes into digital speech.

The Filter/Amplifier

The output of the chip consists of a high frequency pulse width modulated signal which must be passed through a low pass filter in order to remove the high modulation frequency and obtain an analogue signal. This circuit uses a second order Butterworth filter with a cutoff frequency of 5KHz and is made from a CA3140 MOSFET operational amplifier. The signal from this is then amplified to drive a loudspeaker by an LM380 power amplifier in a standard configuration.

Allophones

This type of speech synthesiser utilises parts of the spoken word known as allophones. These are the actual sounds that go to make up speech. The synthesiser board will pronounce fifty nine allophones and in theory it should be possible to synthesise any word in the English language. Obviously though the words produced, while being understandable, will not match those produced by the human vocal tract, which has the capability of producing many more than fifty nine speech sounds!

Conversion of text to speech using allophones requires a small amount of experimentation in order to produce realistic sounds. The amount of effort depends on the composition of the word: Words with nasal and fricative sounds are harder to set right than words that contain mostly just vowels and consonants, but this is just a general guide and it is worth remembering that the sounds of different groups of letters change depending on their position within a word, and that some groups of letters have quite complicated allophone combinations. The allophone table **(Fig 2)** gives allophone numbers in decimal and a guide to their use and should help in the formation of allophone speech.

Programming The Synthesiser

The synthesiser takes a six bit code representing an allophone and generates the corresponding sound. The actual process of sending allophone codes to the synthesiser is simplicity itself and consists of just waiting for the chip to finish what it was last doing and then sending it the new code. This can be done with the following extract of BASIC code:

```
1000 IF IN 159>127 THEN GO TO
     1000
1010 OUT 159,DATA
```

Thus the following program can be used to test a sequence of allophones:

```
 10 RESTORE 100
 20 IF IN 159>127 THEN GO TO 20
 30 READ N: OUT 159,N
 40 IF N THEN GO TO 20
 50 STOP
100 DATA 42,15,16,9,49,22,13,51,0
```

In this example the word spoken will be "COMPUTER" (KK,AX,MM,PP,YY, UW,TT2,ER in allophones!) but any sequence of codes can be put as data at line 100 terminated by a zero value.

For more examples of words made from allophones (some better than others!) a program will be given next month which prints what it says just in case it is not understood!

Another area of experimentation would be to use the synthesiser to produce sound effects for games, something it could do without much computer intervention as the synthesiser will continue a sound until it receives a pause code. The next step up from the example programs would be one that converts allophone input as text into numeric strings to be sent to the synthesiser – this would go some way towards simplifying the text to allophone conversion process. Then perhaps a program could be written to convert English text directly into allophone codes.

So to conclude, this synthesiser gives one the opportunity to add speech to almost any program very cheaply. Let your computer answer back!

Next Month: PCB and Software.

PARTS LIST

Resistors	
R1	10k
R2, 7, 8	330R
R3, 4	100R
R5, 6	1k0
R9	10k
R10	18k
RV1	5k
Capacitors	
C1	10u
C2, 5, 9	220u
C3	220p
C4	820p
C6, 7, 8	100n
Semiconductors	
IC1	74LS42
IC2	SPO256-AL2
IC3	7805
IC4	74LS32
IC5	74LS00
IC6	CA3140
IC7	LM380
Miscellaneous	
Speaker, PCB, IC sockets, connecting wire etc.	

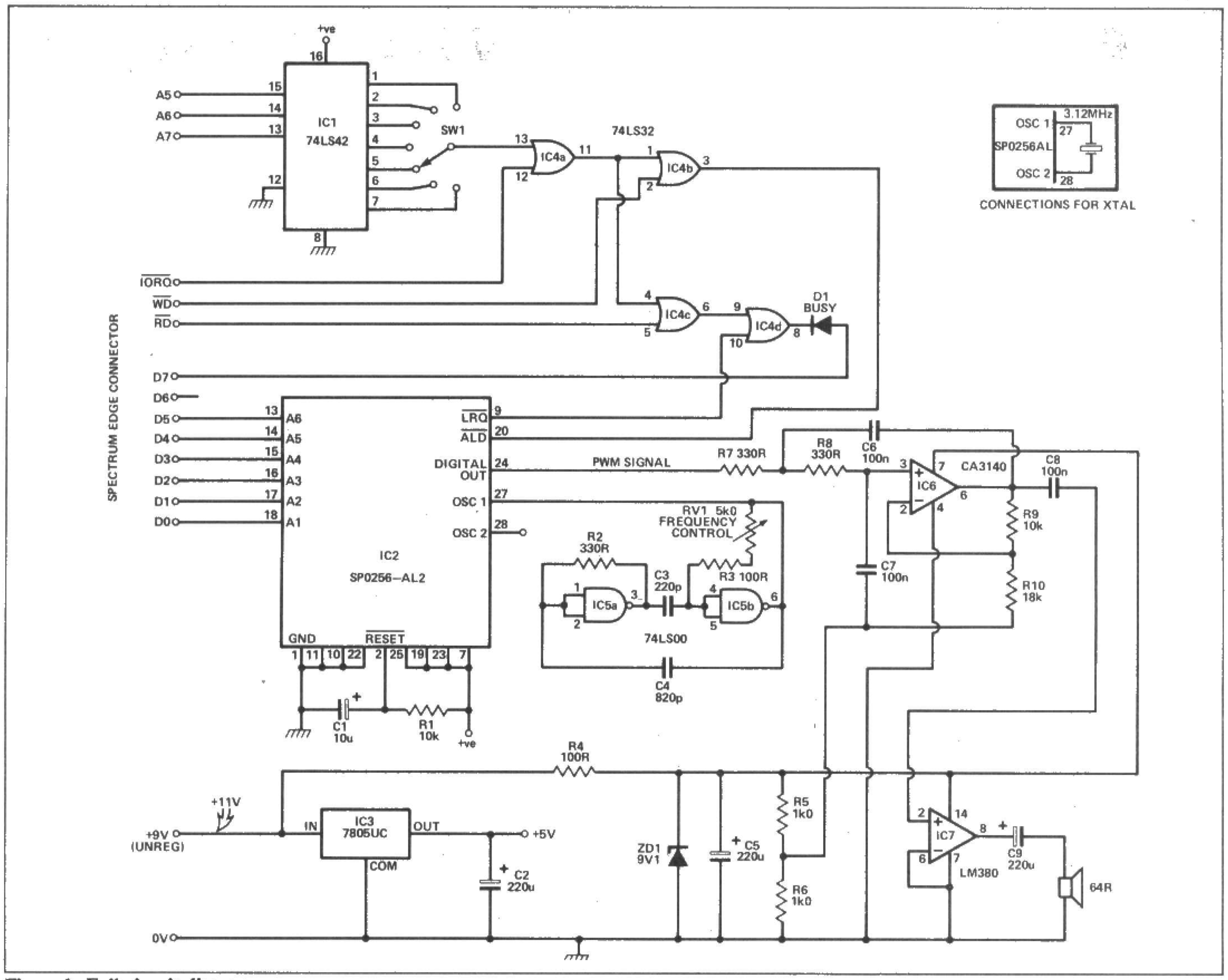


Figure 1. Full circuit diagram.

Allophone Table

Pauses

0	PA1 (10 mS) –	use before voiced stops and affricates
1	PA2 (30 mS) –	use before voiced stops and afficates
2	PA3 (50 mS) –	before voiceless stops and voiced fricatives also between words
3	PA4 (100mS) –	Between clauses and sentences
4	PA5 (200mS) –	Between clauses and sentences

Short Vowels – These can be repeated

7	EH	E	– bend
12	IH	I	– fitting
15	AX	U	– succeed
23	AO	AU	– aught
24	AA	O	– cot
26	AE	A	– fat
30	UH	OO	– cook

Long Vowels

5	OY	OY	– toy
6	AY	Y	– sky
19	IY	E	– see
20	EY	EA	– great
22	UW	O	– to
31	UW2	OO	– food
32	AW	OU	– out
53	OW	OW	– snow
62	EL	L	– angle

R-Coloured Vowels

47	XR	AI	– hair
51	ER	ER	– computer
52	ER2	IR	– bird (monosyllables)
58	OR	OR	– store
59	AR	AR	– farm
60	YR	R	– clear

Resonants

14	RR	R	– read
39	RR2	R	– brain
49	YY	U	– computer
25	YY2	Y	– yes
45	LL	L	– luck
46	WW	W	– wool

Voiced Fricatives

18	DH	TH	– they
54	DH2	TH	– bathe
35	VV	V	– even
43	ZZ	Z	– zoo
38	ZH	GE	– beige

Voiceless Fricatives

29	TH	TH	– thin
40	FF	F	– fire
55	SS	S	– sat
27	HH	H	– he
57	HH2	H	– hoe
37	SH	SH	– shirt
48	WH	WH	– whig

(29, 40, 55 double for initial positions)

Voiced Stops

28	BB	B	– rib
63	BB2	B	– big
21	DD	D	– could
33	DD2	D	– do
36	GG	GU	– guest
61	GG2	G	– go
34	GG3	IG	– wig

Voiceless Stops

17	TT	T	– its
13	TT2	T	– to
42	KK	C	– computer
41	KK2	K	– sky
8	KK3	C	– crane
9	PP	P	– pub

Affricates

10	JH	J	– jury
50	CH	CH	– church

Nasal

16	MM	M	– milk
11	NN	N	– earn
56	NN2	N	– no
44	NG	NG	– bans

THE HIGH RESOLUTION ORIC

The Oric's high resolution mode is unique amongst home micros, but the information contained in the manual is inadequate. S. M. Gee has set out to fill in the gaps and correct some misleading impressions.

The Oric's low resolution graphics and text screen are not difficult to use once you get accustomed to the serial attribute method of controlling the way that characters are displayed. There is certainly no shortage of information on serial attribute low resolution graphics because it is also the method used by teletext and videotex displays. This means that the Oric programmer can look at a range of examples simply by finding a Ceefax, Oracle or Prestel TV set. Also there is large amounts of information on using teletext graphics published in connection with the BBC Micro's Mode 7 graphics.

When it comes to the Oric's high resolution graphics, however, the situation is very different. For although the high resolution mode still uses the serial attribute method, it is unlike any other graphics system. To add to this difficulty the Oric's manual is also short on both information and examples of how colour should be handled in high resolution. Indeed if you read the manual then the few examples given leave the impression that high resolution colour is controlled by POKEing data to screen addresses. In fact it is quite unnecessary to abandon the convenience of the x and y co-ordinate system in either high or low resolution graphics; the Oric has enough graphics commands for all the usual operations found on other machines. In an effort to try and right the balance of information in favour of the Oric's high resolution graphics this article explains both the way that the high resolution display works and how the Oric's BASIC commands can be used to control it.

The High Resolution Memory Map

Although it is not necessary to know anything about the screen's memory map it is interesting and it does help to explain why the Oric's high resolution commands work in the way that they do. The Oric always reserves enough memory to produce a high resolution display even if you have no intention of using it. No matter which mode you are in the structure of the memory use is always the same – first the screen is stored, followed by the alternate character set and then the standard character set (see **Fig 1**). For a low resolution or text screen 1120 bytes of memory are used. For a high resolution screen this increases to 8160 bytes. This implies that when you change from low to high resolution graphics both character set definitions have to be either moved or re-initialised at their new positions. Fortunately, the Oric makes the right choice and moves the existing character set definition so preserving any user defined characters that a program might have produced. Thus if you always define characters while in text mode you only have to remember one address, 46080, as the start of the definitions. This address also applies to the 16K Oric because although the physical location of screen and character definition memory is 32768 lower, hardware makes it respond correctly to the 48K Oric addresses. So, if you want to write programs that work on both the 48K and 16K machines, always use addresses appropriate to the 48K Oric and let the 16K Oric look after itself – notice that the reverse tactic will fail! The only other comment worth making about the graphics memory map is to reiterate the importance of using GRAB if the high resolution graphics are not intended for use. GRAB will allow BASIC programs to use 7K extra memory which can be a significant amount if you are using a 16K Oric. It is important to realise that the effect of GRAB continues until either the Oric is switched off or a RELEASE command is given.

Once the details of the location high resolution graphics screen have been stored in memory, the next question is how the data stored in this area determines what is displayed on the screen. The key to understanding the Oric's high resolution graphics is in realising that each memory location in the screen area controls the state of just six dots, or 'pixels' on the screen. This works in the following way: The least significant six bits, ie. b0 to b5 (see **Fig 3**) of a memory location in the screen area determine the pattern of foreground and background pixels on the screen. A zero bit corresponds to a background pixel and a bit set to one corresponds to a foreground pixel. The first memory location in the screen memory, ie. 40960, controls the first six pixels on the top line of the screen. That is, b5 of memory location 40960 controls pixel 0,0; b4 controls pixel 1,0; and so on to b0 which controls pixel 5,0. The next memory location controls the following six pixels on the same line and so on to the end of the line when the pattern repeats on the next line down. The best way to see how screen memory corresponds to groups of six bits in the display is to set up a high resolution screen and then POKE a fixed pattern of foreground dots into each memory location in turn. If the following program is entered the high resolution memory map can be seen in action.

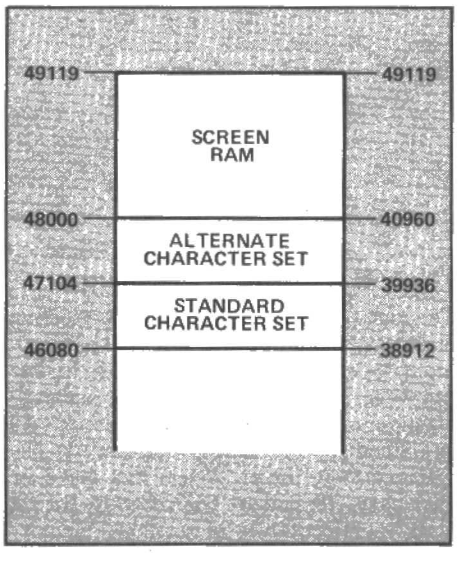


Figure 1. Oric screen RAM.

```
10 HIRES
20 FOR A=40960 TO 49119
30 POKE A,127
40 NEXT A
```

This program POKEs 127, which corresponds to a pattern of all foreground pixels into each screen memory location. As this program runs the screen fills up from left to right and top to bottom. Anyone patient enough to wait for the program to fill the entire screen will find that the last 120 memory locations correspond to the three text lines at the bottom of the screen. Notice that the text lines are controlled in the same way as the full text or low resolution screen: that is, by storing the ASCII code of the character to be displayed in the appropriate memory location. The high resolution screen proper uses memory between 40960 to 48999.

Although the high resolution and text screens use a different display method there is a close similarity between them. The structure of a line of the text screen is composed of 40 character locations. Each character location is composed of eight lines (each of six dots) and is controlled by a single memory location. The ASCII code stored in this memory location is automatically translated to the correct pattern of dots using the character definition tables. Notice that this translation occurs every time the screen is displayed; so changing a character definition changes all of the existing example of that character on the screen. The structure of a single line of the high resolution screen is similar in that it is composed of 40 groups of six pixels, usually called 'character cells' (see **Fig 2**) but, as already described, the dot pattern within each character cell is determined by the bit pattern stored in a single memory location.

The Graphics Commands

Although the Oric manual describes the fundamental graphics commands it doesn't really explain their function or how they should be used together. For this reason it is worth giving a quick summary of each command.

There are only four fundamental high resolution graphics commands: CURSET, CURMOV, DRAW and CIRCLE. To

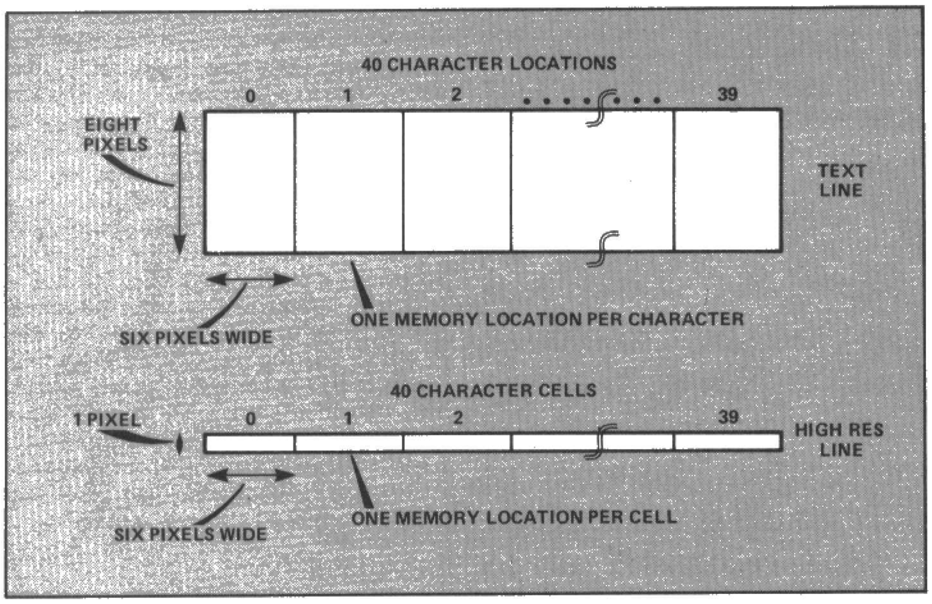


Figure 2. Text and Hi-res.

understand their action it is necessary to know two things. Firstly, the Oric's high resolution graphics mode is based on the use of a graphics cursor. This cursor is moved around the screen using the CURSET, CURMOV and DRAW commands and it fixes the centre of the circle produced by the CIRCLE command. Secondly, a high resolution pixel can only be either a foreground pixel or a background pixel. The colour that a foreground or background pixel actually shows is set by serial attribute codes and this will be described in a later section.

The commands CURSET x,y,fb and CURMOV m,n,fb are most often used to move the cursor round the screen. CURSET x,y,fb will move the cursor to the pixel at x,y. CURMOV m,n,fb will move the cursor m pixels horixontally and n pixels vertically. That is, if the cursor is already at x,y then CURMOV m,n,fb will move the cursor to x+m,y+n. Thus CURSET is an absolute cursor move and CURMOV is a relative cursor move. These commands can also be used to alter the pixel that the cursor is moved to. What actually happens depends on the values of fb, which are as follows:

fb	action
0	set pixel to background
1	set pixel to foreground
2	invert pixel – ie. a background pixel is changed to a foreground pixel and vice versa
3	do nothing

So CURSET 10,5,1 moves the cursor to 10,5 and changes the pixel at this position to a foreground pixel, whereas CURSET 10,5,3 simply moves the cursor to the same place without altering any pixels.

Most of the work in high resolution graphics is achieved by using the DRAW m,n,fb. This is another relative command very like CURMOV, in that if the graphics cursor is at x,y then DRAW m,n,fb will move it to x+m,y+n. The difference is that it will change every pixel on a straight line between the old and new cursor positions. Once again, how the pixels on this line are changed depends on the value of fb. For example, DRAW m,n,1 will draw a line in foreground pixels and DRAW m,n,3 doesn't change any pixels and so is entirely equivalent to CURMOV m,n,3. To draw a line of foreground pixels between the two points x1,y1 and x2,y2 use the compound statement –

```
CURSET x1,y1,1:DRAW x2−x1,
y2−y1,1
```

The CIRCLE command has to be considered as a luxury. CIRCLE r,fb will draw a circle of radius r centered on the current position of the graphics cursor. As always, the effect on the pixels on the circle depends on the value of fb. To draw a circle of foreground pixels centered on x,y and radius r use the compound statement –

```
CURSET x,y,3:CIRCLE r,1
```

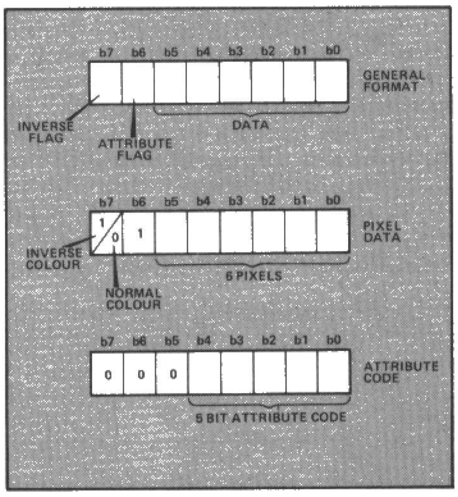


Figure 3. Screen memory formats.

High Resolution Attributes

The high resolution commands described in the last section all work in terms of screen co-ordinates and pixels. What they tend to hide from the user is the way that memory locations determine what is displayed. As explained in the earlier section on the memory map, the six pixels that make up each character cell are controlled by the six least significant bits of a single memory location. This of course raises the question of what the other two bits are used for. In fact b6 and b7 are used as special indicator flags. b7 is the inverse flag and this will be discussed later. b6 is the pattern/attribute flag. When b6 is one the following six bits, ie. b5 to b0, are displayed as the six pixels in the character cell. However, when b6 is zero the lower five bits, ie. b4 to b0, are interpreted as a serial attribute code. These two situations can be seen in **Fig 3.** The high resolution attribute codes are exactly the same as those used to control the text or low resolution screen. However, the only useful codes are:

foreground	**background**	**colour**
0	16	black
1	17	red
2	18	green
3	19	yellow
4	20	blue
5	21	magenta
6	22	cyan
7	23	white

and
attribute code 12 will produce flashing pixels

If a screen memory location is used to hold an attribute code then the six pixels that it normally controls are displayed in the current background colour; this is similar to text serial attribute codes showing up as blanks in the current background colour. The effect that a high resolution serial attribute has is also similar in that it changes the way that all the pixels to its right and on the same line are displayed. So, for example, storing 3 in a screen memory location causes the six pixels that it controls to be displayed in the current background colour and all the foreground pixels to the right of these and on the same line to show as yellow. Of course there is nothing to prevent the storage of more than one serial attribute on a line and so change colours more than once.

The serial attribute method of controlling the screen carries with it the same problems encountered in full colour low resolution graphics. In particular, the fact that attributes show up as six consecutive background pixels severely limits the horizontal resolution. However, at first sight there appears to be an even more awkward problem: there seems to be no way of storing attribute codes on the screen without using POKE, because all the high resolution commands introduced earlier will only change individual pixels, or in other words, the least significant six bits of a screen memory location.

Controlling Attributes With FILL

Once in high resolution mode the two commands INK and PAPER can be used to change the default foreground and background colours in the same way as in low resolution. They work by storing the correct attribute codes in the character cells on the far left of the high resolution screen. However, apart from these two commands, the most useful way of storing attribute codes in the high resolution screen is by using the FILL command.

FILL is introduced in the Oric's manual as a command that can be used to change large rectangular areas of the screen in one operation. Whilst this is indeed one of its uses, the most fundamental and common form of the command is:

FILL 1,1,c

which will store the attribute code c in the memory location controlling the pixel directly beneath the graphics cursor. For example,

CURSET 0,0,3:FILL 1,1,4

will first move the graphics cursor to 0,0 which places it within the six dots that constitute the first character cell on the top line. This causes the FILL command to store attribute code 4 in the memory location controlling the first character cell. The effect on the screen is to display the first six pixels on the top row in the current background colour, ie. black, and any foreground pixels on the top line show as blue. Notice that as it is only necessary for the graphics cursor to be within the character cell for the FILL command to use the memory location that controls it. The command

CURSET x,0,3:FILL 1,1,4

will store attribute code 4 in the first screen memory location if x is in the range 0 to 5. A striking demonstration of this fact can be seen by running:

```
10 HIRES
20 FOR X=0 TO 199
30 CURSET X,X,3
40 FILL 1,1,17
50 NEXT X
```

which moves the graphics cursor one row down and one pixel to the right each time through the FOR loop and uses the FILL command to set a red background. The red moves horizontally across the screen in jumps of six dots, this illustrates the fact that a whole character cell has to be used to store an attribute code, and so colours can only change on character cell boundaries. In this sense high resolution graphics offer no better colour resolution than low resolution graphics, ie. a maximum of only 40 different colour changes can be displayed horizontally. However, high resolution does have the advantage of allowing as many as 200 colour changes vertically. To see this try:

```
10 HIRES
20 C=16
30 FOR Y=0 TO 199
40 CURSET 0,Y,3
50 FILL 1,1,C
60 C=C+1
70 IF C>23 THEN C=16
90 NEXT Y
```

Using FILL it is easy to place attribute codes to set the colour of any point on the screen. If you want to plot a foreground point at x,y in the colour corresponding to attribute code c then use

CURSET x−6,y,3:FILL 1,1,c:CURSET x,y,1

which places attribute code c in the character cell immediately to the left of the pixel at x,y. This method is simple but of course if an attempt is made to plot two pixels less than 12 units apart, the storing of attribute codes will interfere with the display of one of the two foreground pixels. For example, if the above method is used to set the pixel at 50,50 to red and then the pixel at 56,50 to red, the second attribute code will erase the pixel at 50,50! What is less obvious is that Oric high resolution commands will not even attempt to change pixels within a character cell that contains an attribute code. All in all placing attribute codes on the screen is easy. It's where to place them that is difficult!

Of course the complete version of the FILL command is:

FILL row,col,c

and this will store the attribute code c in the character cells forming the block 'row' wide and 'col' deep. The position of the block of character cells is controlled by the graphics cursor in that it is always within the cell in the top lefthand corner.

Inverse Colour

One of the least publicised and most useful features of Oric graphics is the presence of inverse colours. b7 of a screen memory location acts as an inverse colour flag (see **Fig 3**). When b7 is zero the pixels controlled by that memory location are displayed in the current foreground and background colours. However, if b7 is set to one, then the pixels controlled by the memory location display in inverse colours. The inverse of the colour corresponding to attribute code c (c in the range 0 to 7) is simply the colour corresponding to attribute code 7-c. Thus the inverse of black is white, the inverse of red is cyan and so on. The importance of inverse colours is that the foreground and background colours can be changed within a single character cell without having to store an attribute code in the adjacent character cell, and without affecting any other pixels on the same line. The trouble is that the change in colour is very limited, and both the background and foregound colours are changed. To see inverse colour in action try the following program:

```
10 HIRES
20 CURSET 72,50,3
30 FILL 10,1,192
40 CURSET 74,50,3
50 DRAW 0,10,1
60 INPUT I,P
70 INK I
80 PAPER P
90 GOTO 60
```

This sets up a block of inverse colour attributes at 72,50 and then draws a line through the middle, illustrating the effect of inverse colours by inputting different values for INK and PAPER. The main function of inverse colour is to produce large or small blocks of colour that are not surrounded by zones of attribute codes.

User-Defined Characters In Hi-Res

Although user-defined graphics characters are the mainstay of low resolution graphics they are also too good to ignore in high resolution. The command

CHAR a,s,fb

will display the character whose ASCII code is 'a', from the standard character set if 's' is zero and the alternative character set if 's' is one. The character is positioned so that the current position of the graphics cursor marks the top left hand corner of the six by eight character grid. The way in which the foreground points in the character definition affect the screen pixels depends upon the value of fb in the usual way. As an example of the use of the CHAR function consider the following subroutine, which will print the message stored in A$ on the screen so that the top left hand corner of the first letter is at X,Y.

```
1000 CURSET X,Y,3
1010 FOR I=I TO LEN(A$)
1020 CHAR ASC(MID$(A$,I,1)),0,1
1030 CURMOV 6,0,3
1040 NEXT I
1050 RETURN
```

Notice the use of CURMOV at line 1030 to move the graphics cursor on by 6 for each letter that is displayed.

A more interesting problem is controlling the colour of a character placed on the screen using CHAR. In low resolution graphics all that is required to change, say, the foreground colour of a character is the presence of an appropriate attribute code to the left of the character. In high resolution graphics the same principle applies, but eight attribute codes have to be stored – one for each row of the character grid. For example:

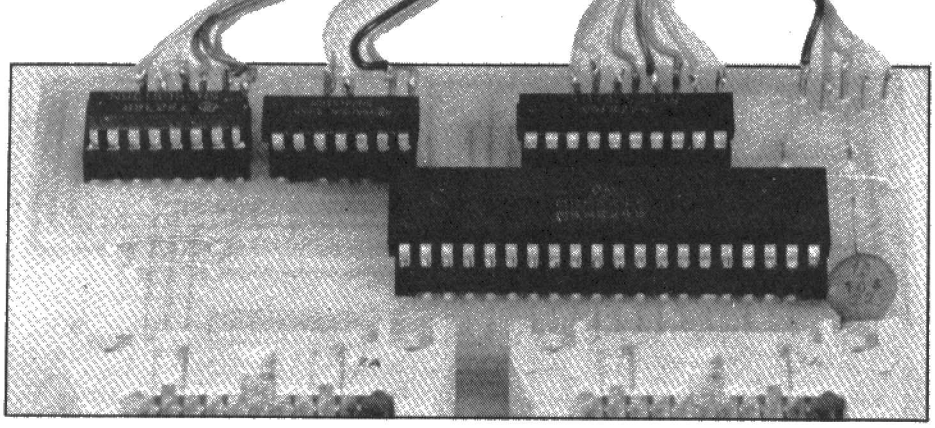
```
10 HIRES
20 CURSET 50,50,3
30 CHAR ASC("A"),0,1
40 INPUT C
50 CURSET 50-6,50,3
60 FILL 8,1,C
70 GOTO 40
```

This program displays the character A on the screen at 50,50 and then FILLs the eight character cells to the left with attribute code C. Using this program the effect of different foreground and background attribute codes can be seen. The main function of user-defined characters in high resolution graphics is to produce small and smoothly moving shapes (see Mike James' series on Micro Graphics in *E&CM* October, November, December '83).

The Oric Challenge

Using Oric high resolution graphics in two colours is easy; the real challenge is in producing multi-colour displays using the serial attribute method. Some displays are easier to produce than might be expected, in that the different coloured areas are well separated. The most difficult type of displays to produce are those which attempt to be general. For example, producing a number of graphs each in a different colour at the same time is virtually impossible without enforcing a number of restrictions such as the graphs not crossing or even coming within 6 pixels horizontally. These problems are fundamental to the serial attribute method used by the Oric. Hopefully, the information given in this article will help you to control the high resolution screen using nothing but BASIC and so free you to concentrate on the more creative use of serial attributes.

E&CM



ELECTRON I/O PORT

Second in the series by R. A. Penfold to upgrade the Acorn Electron to the standard of the BBC Micro.

One of the deficiencies of the Electron when compared to the BBC model B machine is its lack of a user port. However, it is quite easy to add a peripheral interface adaptor (PIA) to the Electron, and the project described in this article is a twin input/output port based on a 6821 device. This provides two 8 bit ports with each line of both ports individually programmable as an input or an output. In addition, each port has two handshake lines, giving a total of twenty input/output lines. The outputs have a built-in latching action, and they can therefore be used to drive LEDs, relays, opto-isolators, etc. with little or not extra circuitry.

The Circuit

Refer to **Fig 1** for the full circuit diagram of the Electron I/O Port.

Address decoding is provided by IC1 and IC2, but only the upper eight address lines are decoded. IC1 is a 4 input NAND gate which gives a negative enable pulse to IC2 when address lines A12 to A15 are all high. The 74LS20 is actually a dual 4 input NAND gate, but in this circuit only one section of the device is utilised. IC2 is a 3 to 8 line decoder, and its three address inputs plus a negative enable input are used to decode A8 to A11. Provided an enable pulse is received from IC1, a negative pulse is produced from pin 12 (output 3) of IC2 when A8 plus A9 are low, and A10 plus A11 are high.

In other words, any address from &FC00 to &FCFF will give a negative pulse from output 3 of IC2. This address range is within the input/output area of the Electron's memory map, but seems to be totally unused by the internal circuits of the machine. Incidentally, the Electron seems to have a rather different input/output memory map arrangement to the BBC model B microcomputer. The BBC machine has a 6522 VIA at addresses from &FE60 to &FE6F, and for software compatibility it might seem the obvious choice to add an Electron user port in this same address range. However, this is not feasible as internal hardware of the Electron seems to occupy at least some of these addresses. For example, ?&FE60 is normally at a value of 176, and writing any other number to this address tends to "crash" the computer! The I/O port has therefore been placed in the unused page &FC.

IC3 is an octal transceiver which is placed between the Electron's data bus and the data bus of the 6821 (IC4). The 6821 is unable to drive the Electron's data bus reliably, and so IC3 is used to provide the necessary buffering. As the data bus is bidirectional a transceiver rather than a buffer has to be used, and the send/receive terminal (pin 1) is controlled by the read/write line so that data is passed in the correct direction. This line is also fed to the appropriate input of IC4.

The negative chip enable inputs of IC3 and IC4 are fed from output 3 of IC2, and both chips are activated when any address in page &FC appears on the address bus. The two positive chip enable inputs of IC4 are not required, and are tied to the positive supply rail. IC4 has six internal registers, but these are selected using just two register select inputs. These are fed from address lines A0 and A1, and the 6821 therefore resides in the memory map from ?&FC00 to ?&FC03. As only partial address decoding is used, the port actually occupies all the addresses in page &FC, with each register appearing at several addresses, but the ones mentioned above are probably the most convenient to use.

The negative reset input of IC4 is simply fed from the corresponding output of the Electron's expansion bus.

The somewhat confusingly named "Enable" input of the 6821 would normally be fed with the 1MHz clock which acts as a

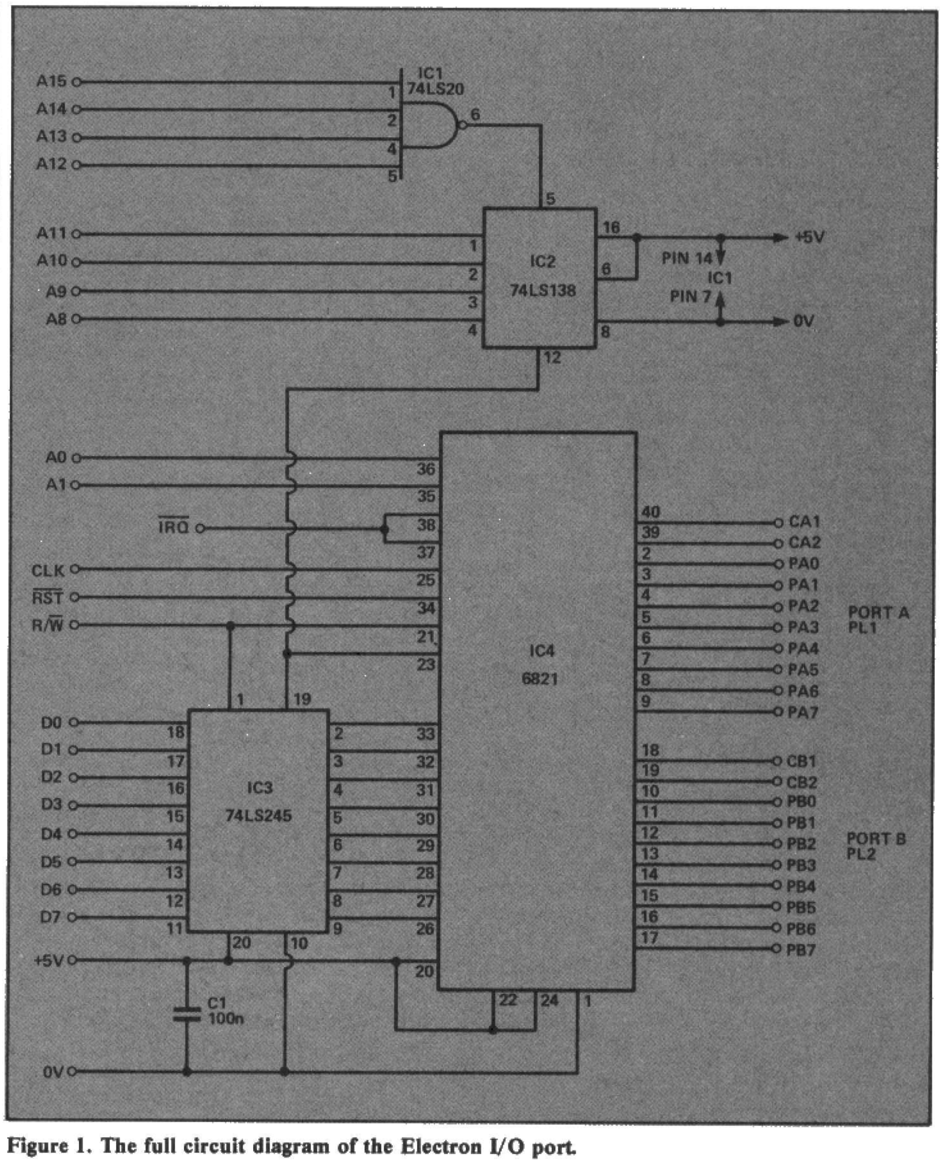


Figure 1. The full circuit diagram of the Electron I/O port.

timing signal during write operations. The Electron does not use a straightforward 1MHz clock frequency, mainly due to the contortions that are necessitated by using four 64K x 1 bit RAM chips to provide 32K x 8 bits of RAM. There are three clock signals available on the expansion bus: 1MHz (approx.), 16MHz, and the microprocessor clock (which is not at a fixed frequency).

Results using the 1MHz clock are rather unreliable, but perfect reliability seems to be obtained using the microprocessor clock, or even the 16MHz clock. Either solution is acceptable when using a 6821, but with a more complex interface device such as a 6522 VIA neither signal would give satisfactory operation of the counter/timers, and some way of using the 1MHz clock would then have to be found.

The 6821 can generate interrupts using the handshake lines, provided the latter are set up for the correct mode of operation. There are two interrupt outputs (IRQA and IRQB); these would normally be wired together and connected to the interrupt request (IRQ) input of the computer. The IRQ outputs are connected together and made available on the printed circuit board, but it is probably best not to connect this output to the IRQ input of the Electron unless you have a specialised application which requires the use of interrupts. This avoids the possibility of generating accidental interrupts and "crashing" the computer.

The two ports are designated port A and port B by the 6821 manufacturers, and they are similar, but not identical. Both are TTL compatible, but port A also has CMOS drive capability.

Power for the unit is obtained from the Electron, and this is quite acceptable as the current consumption is quite modest.

Construction

Details of the printed circuit board are provided in **Fig 2.** Start by fitting the link wires and C1. Then fit IC1 to IC3 and the 40 pin DIL socket for IC4. Note that the integrated sockets do not have the same orientation, and be careful to fit them onto the board the right way round. Next, the connectors for the two ports are added. These are both 20 pin IDC plugs having right angle printed circuit pins for horizontal mounting. When fitting these there is inevitably a great many connections to be made within a small area of board, and a careful check for accidental short circuits due to excess solder should be made.

The board is connected to the expansion bus of the Electron using a piece of 23-way ribbon cable. It is unlikely that 23-way cable will be available, but it is quite easy to split off a suitable piece from (say) a length of 26-way cable. It is easier to connect the lead to the board via Veropins rather than direct, but be careful not to produce short circuits between adjacent pins. If necessary, use pieces of PVC sleeving over connections to ensure that short circuits cannot occur.

The other end of the cable connects to a 2 x 25-way 0.1 inch edge connector, which in turn fits onto the rear of the Electron. There is a slot in the Electron's printed circuit board which will take a polarising key in the edge connector, so that the latter cannot be connected upside-down. At present an edge connector fitted with a suitable polarising key does not seem to be available, but with a little ingenuity it should be possible to add a suitable key. Alternatively, the top and bottom of the connector can be clearly marked as such.

To make things as straightforward as possible the connections on the board have been arranged to correspond to the order in which the connections are made to the edge connector. Refer to **Fig 3** for details of the expansion bus. This shows the connections as viewed looking onto the rear of the Electron, or onto the pins of the edge connector. Again, be careful to avoid accidental short circuits and use sleeving over the connections if necessary.

Finally, plug IC4 into place, and thoroughly check the unit for errors.

Using The Ports

With the unit connected to the Electron, when switched on the latter should work normally. If not, switch off immediately and recheck everything.

Assuming that all is well, the ports are ready for testing and use.

As stated earlier, the 6821 occupies four addresses, but there are six registers. The table given below helps to explain how this system operates.

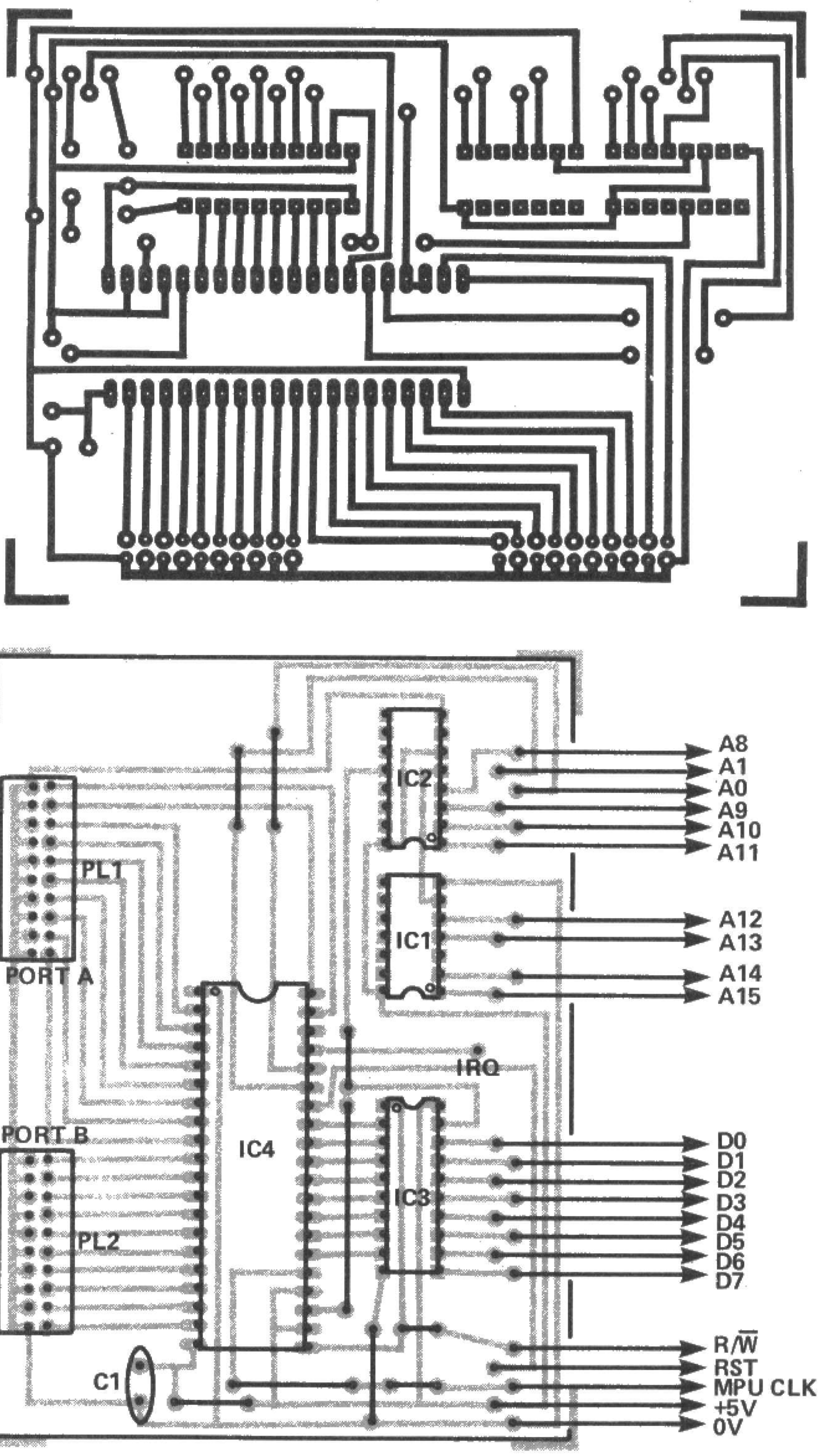


Figure 2. PCB foil pattern, shown actual size (top) and the overlay (bottom).

Address	Control Register Bit 3	Register Selected
?&FC00		Data Direction Register A
?&FC00	0	Peripheral Register A
?&FC01	1	Control Register A
?&FC02	Not Applicable	Data Direction Register B
?&FC02	0	Peripheral Register B
?&FC03	1	Control Register B
	Not Applicable	

Each port has three registers: data direction, peripheral, and control. The data direction register is used to set each line of the port as an input or an output, as required. Writing a 1 to a bit sets the corresponding line as an output, or writing a 0 sets it as an input. For instance, writing 128 to DDR A (all bits at 0 apart from bit 7) sets lines PAO to PA6 as inputs, and PA7 as an output. The peripheral register is the one that is used when reading from or writing to a port. The control registers are primarily used to set the handshake lines to operate in the desired mode and to enable these lines to be read when they act as inputs, but they have another role. With each set of data direction and peripheral registers sharing the same address it is necessary to have some means of selecting whichever of these is required, and bit 2 of the appropriate control register is used to achieve this. Setting this bit low selects the data direction register, while setting it high (in decimal it corresponds to 4 when set high) selects the peripheral register.

Initially all registers are reset to zero, and the data direction registers have all bits at zero. If, for example, we wish to read port A, first control register A at ?&FC01 must be set at 4 (bit 2 high), and then ?&FC00 must be read. It is not essential to first set the control register at 0 and then set the DDR at 0 to set all lines as inputs, since these registers should be set at 0 by the initial reset pulse. However, this can be done to make quite sure that everything is set up correctly. The following would therefore be used to read port A:

```
?&FC01 = 0     (gives access to DDR)
?&FC00 = 0     (sets all lines as inputs)
?&FC01 = 4     (gives access to
                peripheral register)
PRINT ?&FC00(prints returned value)
```

If you try this a value of 255 should be returned since port A has internal pull-up resistors (port B does not, incidentally). Once the port has been set up in the desired mode it is not necessary to repeat this procedure each time it is read or written to, and to read the port again only the last line would need to be used. If you try shorting some of the inputs to the 0V rail and reading the port again, a suitably modified value should be returned. **Fig 4** gives details of the connections to the two ports.

To test port B the following could be used:

```
?&FC03 = 0    (gives access to DDR A)
?&FC02 = 255(sets all lines as outputs)
?&FC03 = 4    (gives access to
               peripheral register B)
?&FC02 = 240(writes 240 to port B)
```

This should set PB0 to PB3 low, and PB4 to PB7 high, and the appropriate states can be confirmed with the aid of a multimeter or logic probe.

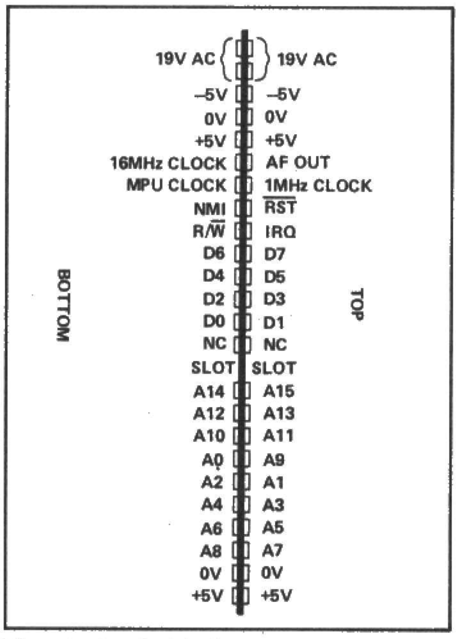


Figure 3. Detail of the Electron's expansion bus.

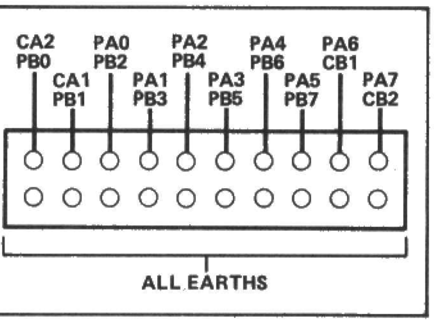


Figure 4. The connections to the two 20-way connectors.

Handshaking

Use of the handshake lines is not difficult to master. The handshake lines of the two ports are used in much the same way, and therefore only the port A handshake lines will be considered here.

CA1 is the more simple of the two handshake lines as it can only be used as an output; it is controlled by bits 0 and 1 of control register A, but with the interrupt output unused only bit 1 is significant. With bit 1 set at 0, CA1 responds to high to low transitions, but with bit 1 set at 1 it responds to low to high transitions. This input only responds to a transition of the appropriate type, and a constant state or transition in the wrong direction have no effect.

An active transition of CA1 sets bit 7 of the control register high. The logic AND function can be used to read just this bit of the register (i.e. PRINT ?&FC01 AND 128 will return 0 if bit 7 is low, or 128 if it is high). Bit 7 of the control register is reset to 0 by reading port A.

To use CA1 as an interrupt input bit 0 of control register A is set high, in addition to setting bit 1 for the desired mode.

CA2 is controlled by bits 3, 4, and 5 of control register A, but when it is used as in input, bit 5 is always 0. Bit 3 is the interrupt enable bit, and is set high if CA2 is to be utilised as an interrupt input. Bit 4 controls CA2 in the same way that CA1 is controlled by bit 1, but an active transition of CA2 sets bit 6 of the control register high. Just this bit of the control register can be read using the logic AND function, and a masking number of 64 (i.e. PRINT ?&FC01 AND 64 returns 0 if bit 6 is low, or 64 if it is high).

When CA2 is used as an output, bit 5 of control register A is always set high. There are four output modes, and these are selected by setting bits 3 and 4 to the appropriate states, as shown in the table below:

BIT 4	BIT 3	MODE
0	0	CA1 Control
0	1	Negative Strobe
1	0	Always Low
1	1	Always High

In the CA1 control mode CA2 is set high by an active transition on CA1, and is reset when port A is read. Note that CB2 operates in a slightly different manner when used in this output mode, in that it is a write operation to port B which resets it.

In the negative strobe mode a brief negative output pulse is produced each time port A is read. Again, CB2 is slightly different when used in this mode, and it is writing to port B which produces the negative strobe pulse.

In the other two modes, bits 4 and 5 are set high, and then CA1 is simply set at whatever state is written to bit 3. This mode is the same for CB2.

As an example of how a control register is set up, let us assume that CA1 is to respond to high to low transitions with interrupts disabled, CA2 is to be used in the strobed output mode, and that access to the peripheral register A is required. This requires bits 1, 2, 3, and 5 to be set high, and in decimal these correspond to 2, 4, 8, and 32 respectively. The total number written to the control register at ?&FC01 would therefore be 46.

E&CM

Next Month:-
Electron RS423 Interface

PARTS LIST

Capacitor

C1	100n disc ceramic

Semiconductors

IC1	74LS20
IC2	74LS138
IC3	74LS245
IC4	6821

Miscellaneous

PL1, 2 20 way IDC sockets, right angle pins (2 off); 23 way ribbon cable; Printed circuit board; 2 x 25 way 0.1 inch edge connector; 40 pin DIL IC socket; 20 pin DIL IC socket; Veropins.

Multi-Tasking Forth For The ZX81

Adam Denning attempts to overcome his prejudice against the FORTH language and reviews a new adaptation of the ZX81 in which the BASIC has been replaced by a FORTH ROM.

The Husband FORTH ROM costs £28.75, and is supplied with a very comprehensive manual.

Ignorance surrounding assembly language has prompted a large section of the computer press to advocate FORTH as an alternative. FORTH is ideally suited to control applications, and being a 'threaded interpretive' language, which means as close to compiled as makes no difference, the speed this language can offer over its major competitors does indeed make it seem almost comparable to assembler. However, there is a large element of subjectivity in this view, and the author not only vastly prefers Z80 machine code to FORTH, but also prefers languages of a higher level such as BCPL and C. Nevertheless there is a *very* strong interest in Charles Moore's reverse Polish control language, and FORTH is well-suited to systems requiring a substantially lower level of control (that is more intimate rather than less complicated) than a higher level language can realistically offer.

The ZX81 maintains an ever odder position in home computing; not even Sinclair have decided if it is actually dead, and now that the £45 starter pack offer has been discontinued, it must surely lose a fair amount of ground. To have a system of any effectiveness whatsoever, the 16K RAMpack is mandatory, and so now one must expect to pay nearer £60 for an outmoded black and white computer with pitiful graphics and no sound. As a home computer it simply fails, its venerable years of service being well and truly over. With the added disincentives of the horrible keyboard and the infamous RAMpack wobble, it is very hard to see the ZX81 in the same light as one could two years ago. A pity yes, but a fact nevertheless. The competition, not least from Sinclair themselves, is just too good.

Why then has David Husband, a renowned FORTH personage, decided to produce a FORTH system for this micro? Perhaps the advantages are with the implementation rather than the computer, this FORTH being in ROM and having the extra bonus of a multi-tasking capability. Initial cynicism on this last point is only partially founded, as Husband has done it well enough to actually make it work.

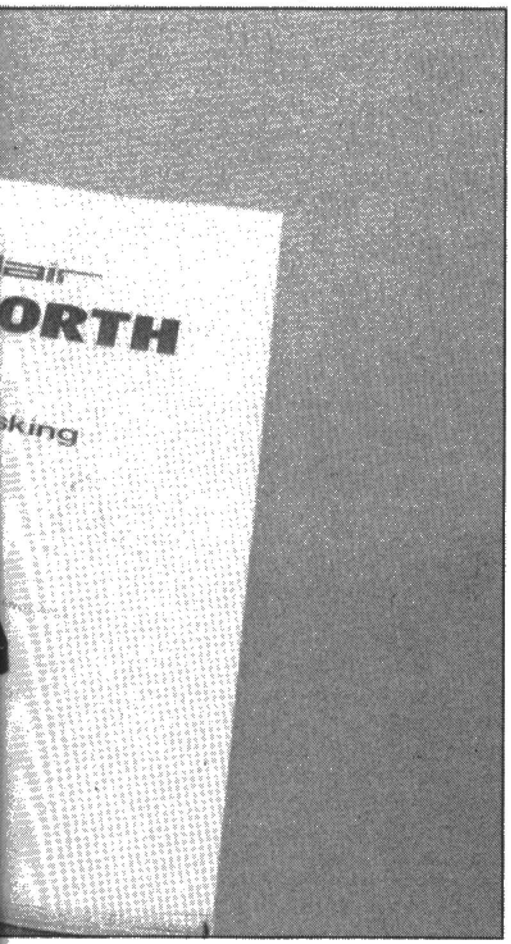
The ROM costs £28.75 (that is, over half the price of the computer itself!), and for this outlay you receive the ROM and a very comprehensive manual for those already well versed in the language. Husband takes the sensible (indeed vital) step of recommending and selling a text book on the subject, "The Complete Forth" by Winfield for £6.95. There are actually better (but more expensive!) books on the subject, but as an initial reader this is as good as any other. Surprisingly perhaps, the Acornsoft FORTH manual (£7.50) is as informed a text as this, although obviously certain features of the book are BBC/Electron specific.

Fitting the ROM into the ZX81 is an easy process for any *E&CM* reader, especially those with later versions of the machine, when it is simply a matter of replacing Sinclair's BASIC ROM and inserting in its place the Husband FORTH ROM. All necessary instructions are provided in the manual, which appears to be a photocopy of the original which was printed using a dot-matrix printer. With 71 extremely well-documented pages this manual is no rush-job, and a thorough read of this is recommended whether you have the ROM or not. You'll learn a lot about FORTH systems and multi-tasking, at the expense of learning virtually nothing about the language itself.

Should the prospect of disemboweling your computer prove somewhat noxious to you, there is an opportunity to buy a ready assembled version of the computer, with the FORTH ROM installed. This would only be a viable option if you were convinced that you had a use for a multi-tasking ZX81, as there is no chance of ever using this converted machine as a standard ZX81; the BASIC ROM is lost forever. In this respect it is odd that upon buying a ready built FORTH ZX81, you are supplied with Sinclair's BASIC manual as well – surely this is redundant and the minute amount of relevant information could have been transferred to the FORTH manual? As a last point about manuals, Husband informs us that he is in the process of writing a full Technical Manual, which will reputedly go into much more detail than the present one, and advanced and/or ambitious users of the system will be able to glean a lot more ROM details from this.

"it is simply a matter of replacing Sinclair's BASIC ROM and inserting Husband's FORTH ROM"

There seems very little point in going into the finer intricacies of FORTH itself as this has been adequately covered elsewhere, not least in the November issue of *E&CM*, so it will be worthwhile to discuss the differences between this implementation and the two standards: Fig-FORTH and FORTH-79. Apart from the FORTH Interest Group (Fig – you get the connection?!), it seems that most people prefer the 79 standard; this version is more comprehensive when it



comes to handling such important programming devices as loops. Nevertheless, the Husband version is based on Fig-FORTH, although limited ROM space (what, 8K limited? – sorry, we forgot the multi tasking. Apologies!) does not allow the full compliment of Fig-FORTH instructions/words/applications (choose your own word from the endless stream of jargon) to be included. To make up for this there is apart from the multi-tasking, (of which a lot more later) a non-standard (and better) compiler/interpreter thingy. Excuse the lack of suitable jargon to adequately describe the way the code is run in one word; what happens is that instead of having a list of addresses for each FORTH word and a corresponding inner interpreter, Husband's FORTH consists of subroutine calls, each subroutine being in native Z80 machine code. Naturally, this is faster, and you would certainly need tests rather more comprehensive than the standard benchmarks to make a comparison with pure machine code. There cannot be that much in it, excepting of course that well-written machine code is by definition optimised, and it is highly unlikely that FORTH words will be to any large degree.

On powering up a thus-installed system, the user is presented with a white screen with a copyright message at the top, and a flashing cursor. This same screen is presented after a COLD restart – effectively, a power down followed by a power up. By simultaneously pressing the SHIFT and EDIT keys, the normal working 'split screen' is displayed – however, as this key combination actually toggles between the two screens in this split format, and as key repeat *is* implemented, holding the keys down for too long (ie. for more than about half a second) switches between the two screens too rapidly, and dextrous maneouvering is necessary to catch the cursor in the right place. Having achieved the split screen, a black block-graphic dividing line appears, with two cursors – the current one flashing, the dormant one not.

From here on in, FORTH advocates will recognise most of the format, with the upper screen being the editor screen. Words compiled on this screen can then be caused to compile and/or execute by issuing the relevant commands to the lower screen. Apart from a couple of bugs involving wrap-around, this screen editor (compared to a real systems screen editor of course, this one does not live up to the name – but this is a ZX81) works as expected, and is perfectly useable. FORTH words and applications can now be written, edited, compiled and executed as per normal, but the only thing that makes this unit at all exceptional is the real time task handling.

To take advantage of this multi-tasking, the real time clock comes into play. Being interrupt driven this clock is at least as accurate as most house clocks, and can cause pre-defined tasks to be executed at pre-determined intervals. You can for instance ask a particular task to start for the first time at say 12 o'clock, and then repeat itself until further notice every two minutes. The actual

"a nice idea implemented on the wrong computer"

time intervals extend from a fiftieth of a second all the way up to a number of years, so if you have an urgent appointment at 7 o'clock on the 23rd June 2023, just get programming! Seriously, considering that by careful task management one can cause say ten different things to be done at regular or even irregular intervals, the real hobbyist should be able to have days of fun with this ROM. Of course, if tasks are due to occur at the same time or if they run into one another, the performance of the system rapidly deteriorates – it goes plain stupid when you configure a task that takes say 2 seconds to execute every second – for obvious reasons.

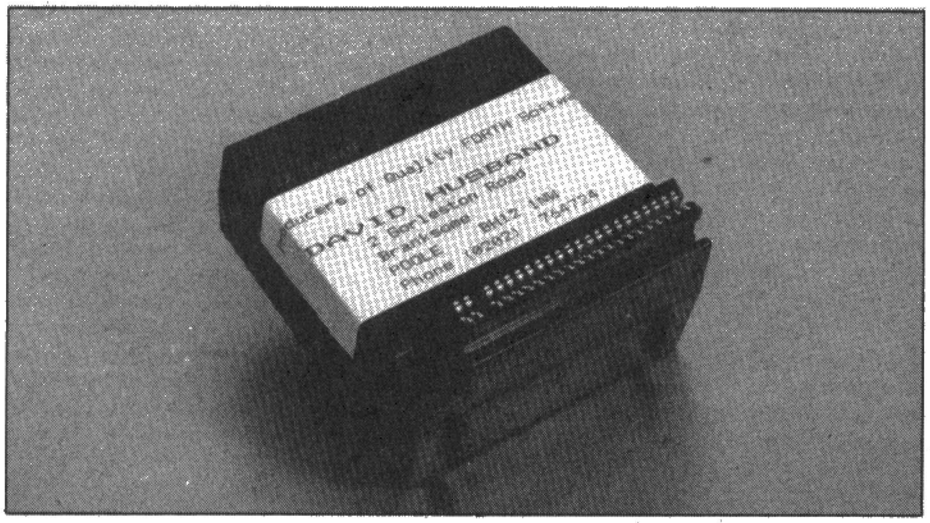
In all then, Husband's FORTH is a very nice idea, but is implemented on entirely the wrong computer. The Spectrum and/or the BBC would have been so much more sensible. Of immediate notice is that despite having a different ROM the ZX81 still suffers from its two main failings from the past – horrible keyboard and extremely frustrating RAMpack wobble.

Mr Husband is however still one step ahead of the pack. He is soon to launch a similar product for the Spectrum which will overcome all the deficiences of FORTH on the ZX81. The new version is in the form of a 12K ROM card, and includes not only the FORTH multi-tasking ROM (and with 12K it must be superior), but also a Z80 assembler in FORTH, a built in terminal emulator that supports a modem (which can also be used from BASIC), a very useful crystal controlled RS232 baud rate generator, a machine code monitor which seems to fit the normal Zilog 'front panel' display, routines for both RS232 and Centronics printers and the spare 4K of ROM space is available to both users and 'coming soon' language extensions. This product then could be described as the ideal solution to all the problems associated with the ZX81 implementation, and being a plug in board rather than a replacement ROM it is also rather more versatile.

The Spectrum unit will be priced at £59, and, considering the inclusion of an 8251 for serial I/O and an 8255 for parallel I/O to say nothing of the comprehensive facilities, this is surely value for money. One is likely to get all the benefits of the ZX81 unit with none of the disadvantages (except FORTH?!!) AND with all those extra facilities, it seems that Husband has got it right this time around.

The ZX81 ROM alone can be obtained from: David Husband, 2 Gorleston Road, Branksome, Poole, Dorset; and the fully-built FORTH ZX81's can be purchased from: Densham Computers Ltd., 329 Ashley Road, Parkstone, Poole, Dorset – at an undisclosed price.

E&CM



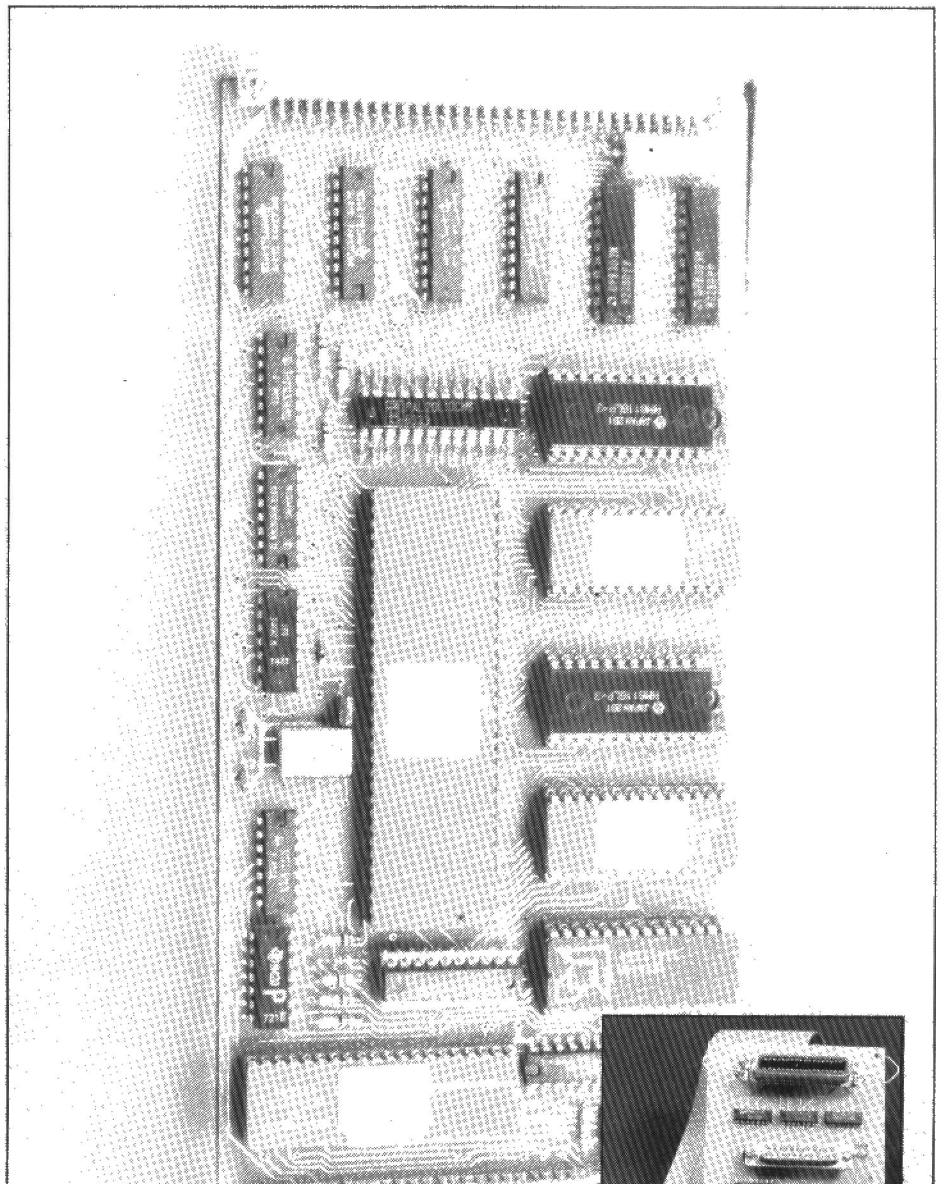
David Husband's 16K FORTH ROM pack.

SAT 16

PART 2

This month our 16 bit computer series, presented in conjunction with Satellite Services, concentrates on the system's circuit diagram.

The circuit for the SAT-16 MPU Card can be separated into two major sections (see the block diagram shown in **Fig 1**) – the 68000 processor with its resources **(Fig 2)** and the 68701 processor with its timing and interfacing facilities **(Fig 3).** The block diagram illustrates the two parts and their interconnection.



68000 Circuit Description

Memory

On-board memory is allocated 2 blocks in the address map by decoding outputs from the PAL16L8 (U19). The 2 EPROM chips are allocated areas addressed 000000-003FFF by signal ROME. Only one common enable line for both upper and lower bytes is required since instruction fetches always transfer one word to the CPU. The two RAM chips are allocated addresses 004000-007FFF by signals RAME1 and RAME2.

Various memory chip types are catered for and the addressing for the different types are broken down as follows:

EPROMs

4K versions (2732) – normal address area = 000000-001FFF. This is duplicated at 2000-3FFF.

8K versions (2764) – normal address area = 000000-003FFF.

16K versions (27128) – this fills the allocated area address space 000000-003FFF. (Lower 8K) and a further space 7F8000-7FBFFF (upper 8K).

EPROMS should be 350 ns or faster. No hardware changes are required to accommodate the different EPROM sizes.

RAMs

2K versions (6116) – normal address area = 004000-004FFF. This is duplicated at 5000-5FFF, 6000-6FFF; 7000-7FFF. (LK6 fitted).

8K versions (5564) – this fills the allocated address space 004000-007FFF. (LK5 fitted).

When any address within the above defined on-board area is accessed the PAL16L8 (U19) will also generate LDTK to complete the asynchronous handshake and another signal called VALID which is used by the PAL20L10 to hold the bus buffers U2-U7 disabled thus ensuring no conflict occurs on reads between data coming from the on-board memory and data from the G64 bus.

68K Interrupts

The 68000 has provision for external interrupts via pins IPL0-2. There are 8 possible combinations of these 3 lines but only 7 levels of interrupt are allowed since a logic low on all 3 lines means no interrupt. This fact is utilised by the LS348 priority interrupt encoder. IN0 to the LS348 is permanently grounded and therefore in the absence of any other interrupt line input to the LS348 the 3 outputs are held in a low state. This means that the highest priority interrupt line that is asserted low to the LS348 will be encoded on to the 68000 IPL lines.

The 68000 also allows provision for external interrupt requests to be auto-vectored (ie. the software is switched automatically to a pre-defined handling routine depending on the decoded interrupt level) or user vector where the interrupting device has to provide an 8-bit vector number in response to an interrupt acknowledge from the CPU. Auto and non-auto interrupt are pre-defined on the SAT-16 MPU Card by logic contained within the PAL20L10 (U8). The internal logic of this PAL is programmed such that all external interrupts except INT6 are auto-vectored by asserting VPA back to the 68000 for these levels during an interrupt acknowledge cycle. This is not done for a response to INT6. Instead the 68000 will expect a vector number on the lower 8 data lines with a DTACK reply when it acknowledges a level 6 interrupt (see the relevant 68000 data sheet for more details of the interrupt sequence). Note that all interrupts up to and including INT6 may be masked off within the 68000 by software but INT7 is non-maskable and will always be responded to as auto-vectored. Interrupt acknowledge (IACK) is decoded from the 68000 function code lines FC0-2 by the PAL20L10 and is sent to the G64 bus via an inverter (U18) as well as being used internally by the PAL. INTs 7, 6 and 5 are available to the user on the G64 bus. INT4 is not used. INTs 3, 2 and 1 are used by on-board logic as handshake and control between the 68000 and 68701.

Bus Addressing

The standard 68000 CPU is capable of directly addressing 16 Mbytes of memory. The SAT-16 MPU system allows for access of up to 1 Mbyte asynchronous addresses in a continuous block and a separate area for synchronous devices. The normal G64 bus specification gives 17 address lines which in a 16-bit environment are word addresses not byte addresses. Upper and lower data strobes qualify the two 8-bit groups of data lines for transfer purposes. Two extra address lines are available from the SAT-16 MPU Card by fitting links for addresses 17 and 18. THESE LINKS MUST NOT BE FITTED unless the user is utilising the expanded version of the G64 bus (ie. pins 1A and 1B are bus lines and not ground). With these links fitted and utilising the expanded G64 bus, 19 address lines become available giving an address capability of 512 K words or 1 Mbyte. There are also two address strobe lines available on the G64 bus (AS and VPA). These are selected by using A23 on the 68000 and are enabled in the following manner: If software attempts to access an address with A23 low it is assumed to be a normal memory address and AS is asserted. This starts an asynchronous transfer and the CPU expects DTACK to complete the cycle. If software attempts to access an address with A23 high then the PAL20L10 (U8) detects this and signals to the 68000 that a synchronous transfer is required. The 68000 then synchronises its internal timing and generates a 6800-type cycle using VMA (VPA on the G64 bus) and ENABLE to qualify the transfer. Due to on-card resource requirements, not all of the possible addresses are available to the CPU from the G64 bus.

Addressing Continued

The address block 000000-007FFF and 7F8000-7FFFFF is used by on-card memory and any accesses attempted to addresses within this range will inhibit any signals from the SAT-16 MPU Card to the bus (U2-7 are tri-stated). Also addresses in the range 808000 upwards are not available since this block is reserved for I/O and future SAT-SYSTEMS enhancement. These blocks are detected by the PAL16L8 (U19) and generate the VALID signal to the PAL20L10 which inhibits bus buffers U2-U7. This restriction does not mean that devices within these address ranges may not be used on the G64 bus, it only means that the SAT-16 MPU does not have access to them. These addresses may be used by other devices or processors on the G64 bus for other purposes such as DMA buffers for peripheral to peripheral transfers.

SAT-16 MPU designers recommend that a watchdog timer be used on the G64 bus structure that monitors the AS and DTACK lines on the bus. This watchdog is required to assert the BERR line low if an AS has been detected with no responding DTACK within 10 full cycles of SYCK. This delay figure is recommended since planned future devices utilising the G64 bus structure may require up to 8 SYCK cycles to respond. If the 68000 detects BERR in response to a transfer attempt, then it will vector to a handling routine to take appropriate action. If the SAT-16 MPU Card is used as a stand-alone system then the user MUST connect AS to BERR on the G64 connector. In this instance a watchdog timer is NOT required.

Error Detection

Under certain conditions the 68000 may enter into a state whereby it can no longer function (eg. double BERR). In this condition the 68000 will generate a low level on the HALT line. Logic internal to the PAL20L10 monitors the HALT line to the 68000 and the HALT line from the G64 bus and is capable of determining whether a HALT is generated from the 68000 as a result of an error condition, or whether it is as a result of a request from the bus. If an error condition is detected the PAL20L10 (U8) will generate a signal (CRASH) to the 68701 processor which is handled as a NON-MASKABLE INTERRUPT by that processor. This means that since the 68701 controls the interface to the USER TERMINAL it is possible to at least inform the user that the 68000 can no longer function rather than having everything 'hang up' leaving the user wondering if anything is still running, as might be the case with most systems without this facility.

Reset

The SAT-16 MPU Card has 2 sources of SYSTEM RESET. The first is a POWER ON RESET implemented by a capacitor charge circuit (C3,R5) driving a Schmitt triggeral inverter (U18). This gives a long enough time constant after POWER-ON to enable all the system components to stabilise. To effect a clean RESET to the 68000 CPU the manufacturer requires that both RESET and HALT be asserted low together. This is accomplished by the two inverters U16 pins, 1,2,3,4. The PRESET line from U16 pin 4 as well as providing the RESET input to the 68000 also provides the RESET to the G64 bus (pin 14b) and the 68701. This particular line is actually OPEN-COLLECTOR from U16 so that the RESET INSTRUCTION facility of the 68000 instruction set may be utilised to other devices on the G64 bus.

The second source of system reset is from a switch connected, via the adaptor connector J2 pins 37, 39, 40, to U17 pins 2 & 5. These two gates debounce the switch contacts and via U16 pins 5 & 6 will discharge the POWER-ON RESET timing capacitor (C3). Thereafter the RESET follows the normal POWER-ON RESET sequence.

68701 Circuit Description

The 68701 is a versatile micro-computer chip. It is not within the scope of this series of articles to give a detailed functional description of the internal workings of this device. Therefore we will only describe its functions as related to its role on the SAT-16 MPU Card. For an in-depth appreciation of this chip it is recommended that the reader

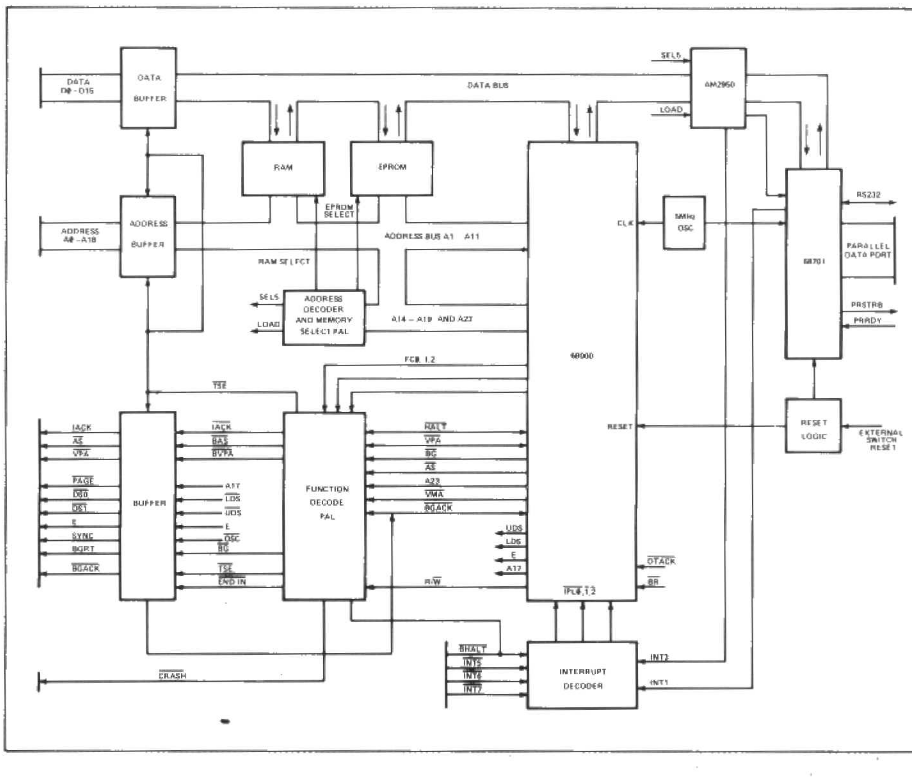


Figure 1. Block diagram of the SAT 16 system.

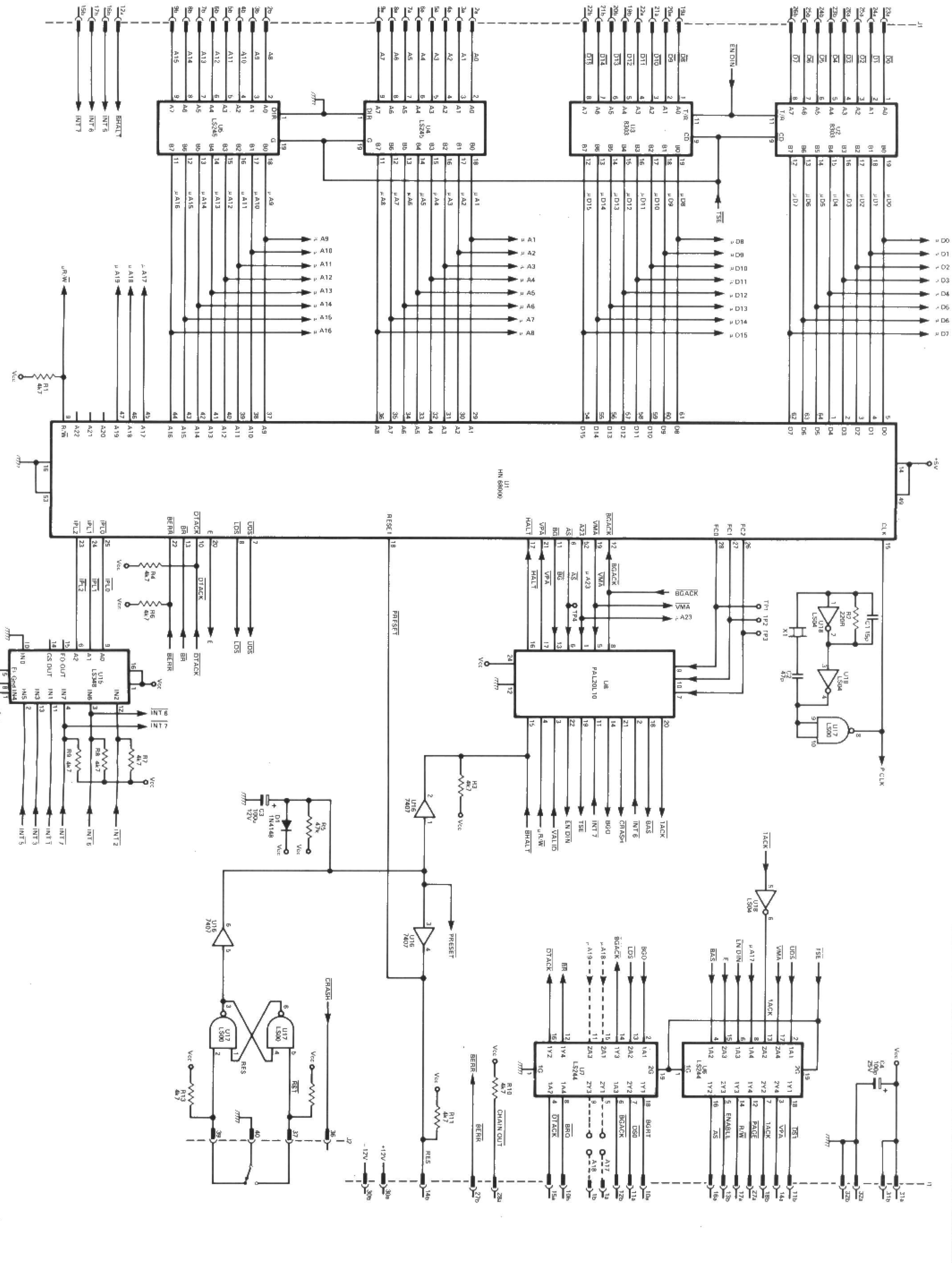


Figure 2. Circuit diagram of the 68000 and its resources.

Kits and monitor software available from Satellite Services – see their advert in this issue.

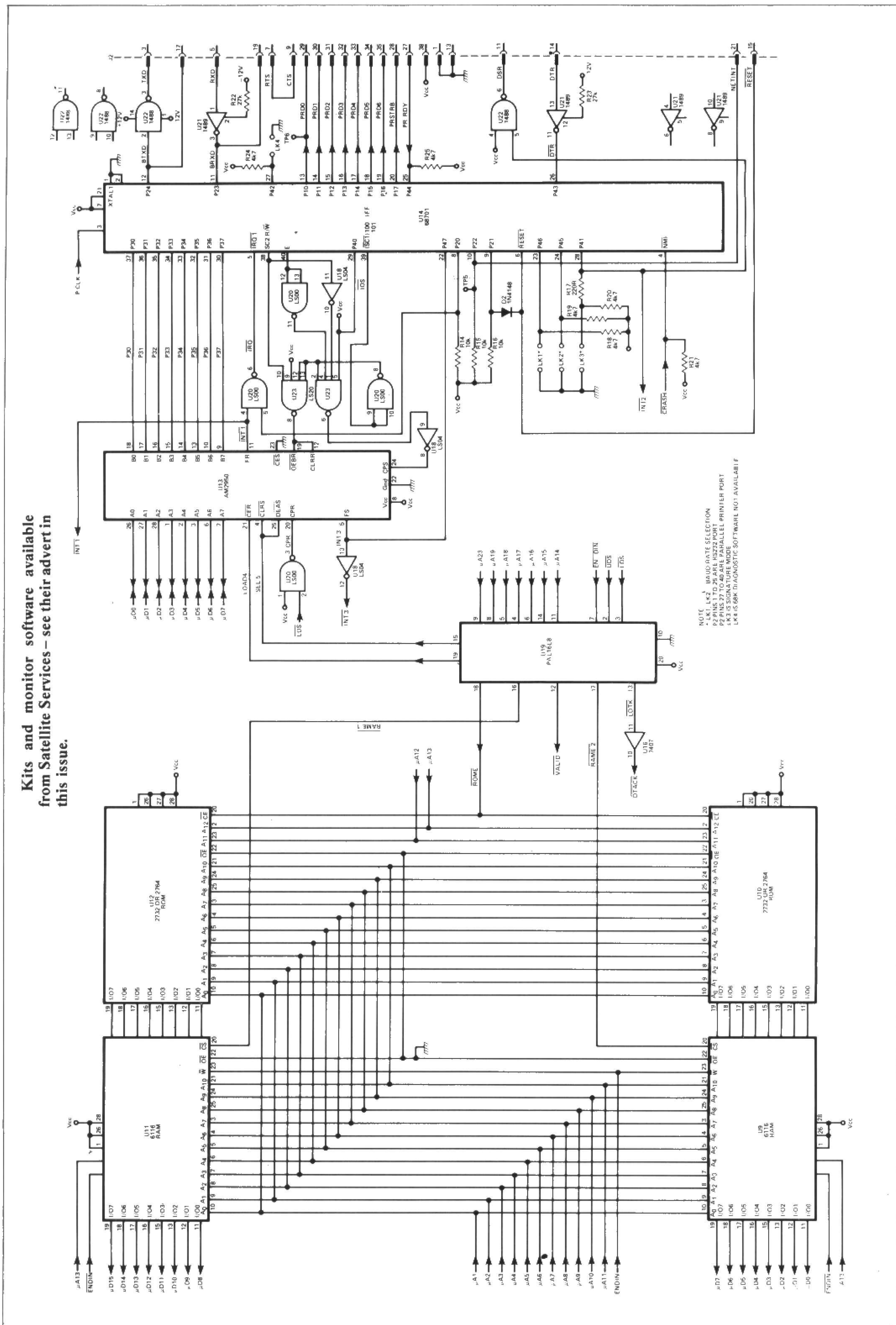


Figure 3. Circuit diagram of the 68701 with its timing and interfacing facilities.

obtain the manufacturer's data sheet to fully understand some of the references made in the following description.

The chip is used in its expanded non-multiplexed Mode (5). Port 1 lines 0-6 are used as a 7-bit parallel interface to a printer, therefore providing ASCII codes (00-7F). Port 1 bit 7 provides the printer strobe pulse with the acknowledge returning on Port 2 bit 2 and PRINTER READY status on Port 4 bit 4 line. Port 2 lines 3 & 4 are configured as a serial interface to the USER TERMINAL with bit 4 being TX data out and bit 3, RX data in. The baud rate setting for the serial link is set up by reading links LK1 & LK2 during the RESET sequence. The settings of these two links select one of 4 preset internal divider ratios which use the CPU Clock to derive the 16x bit rate signal required internally for the SCI. DSR is provided to the USER TERMINAL when power is applied to the card and 'signature mode' is NOT selected. DTR input from the terminal is read, at reset time, from Port 4 bit 3 by the processor. All the above-mentioned lines are available on connector J2. Port 3 is used as a parallel data link between this processor and the 68000. A more detailed description of this interface is covered in the next section.

RESET comes in from the previously-mentioned RESET CIRCUITRY and, as well as performing a reset to the processor, it also sets the operating mode to Mode 5 via diode D2. NMI on pin 4 is generated from PAL20L10 (U8) as a result of a 68000 crash as previously mentioned. IRQ on pin 5 will be covered in the next section.

68701 – 68K Link

The 2950

The 2950 (U13) is a very useful device for interconnecting two computers or bus systems. It is a bi-directional tri-state 8-bit buffer latch with full hand-shaking logic in both directions. The role of the 2950 in data transfers is quite complex and will be described in two parts:

1) Transfers from 68701 to 68000:

Internally to the 2950 is an 8-bit register designated the 'S' register. This takes its input from the 'B' interface and data is clocked into it by a positive going edge on CPS (pin 24). This pulse on CPS also sets an internal F/F, FS to indicate that data has been loaded into register S. The output of this F/F, FS (pin 5) asserts an interrupt to the 68000 CPU (INT3) to indicate that new data is waiting from the 68701. Output FS is also fed back to the 68701 on Port 4 bit 7 so that the 68701 can tell when the 68000 is ready to accept the next character. The 2950 S register is tri-state controlled by OEAS (pin 25). When the 68000 is ready to take the byte of data from the S register it performs a 'READ' to the 2950 memory address which is decoded by the logic in PAL16L8 (U19) to generate the signal SEL5. This enables the S register data onto the 'A' interface where it is taken by the 68000 D0-D7 lines. SEL5 is also connected to CLRS (pin 4) which will reset the S F/F on its trailing edge thus informing the 68701 that the buffer is ready for the next byte of data.

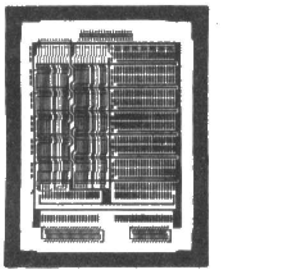
2) Transfers from 68000 to 68701:

These are done in a similar manner to the 68701 to 68000 transfers only this time another internal register called 'R' is used with its own associated F/F, FR. The 68000 writes a byte of data on D0-D7 to the 2950 address decoded by PAL16L8 to give the LOAD4 signal. At LDS time this data is latched into the R register and FR is set. FR output (pin 11) generates an IRQ to the 68701 and is also fed back to the 68000 as INT1. The 68701 is capable of externally masking off this interrupt via a low on Port 2 bit 0 if it is in a condition where it is not ready to accept data. When the 68701 is ready it can read the data from the R register by performing a READ to the 2950 address, decoded by U20, U18, U23, which puts a low strobe to OEBR (pin 19). This allows the R register data onto the 'B' interface. This enable is also connected to CLRR (pin 12) so that on its trailing edge it resets FR, thus asserting INT1 back to the 68000 to inform it that the buffer is available for another byte transfer.

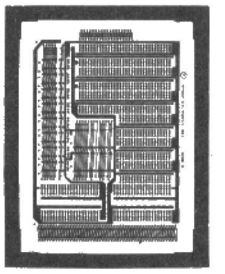
CPU Clock

The clocks for the 68000 and the 68701 are generated by two separate crystal controlled oscillators detailed as follows – 68000 CPU clock: the clock for the 68000 is generated by a crystal controlled oscillator made up of IC24 and XTAL 1 which gives a frequency of 8MHz. 68701 CPU clock: the 68701 has an onboard oscillator controlled by an external 2.5MHz crystal which also provides the clock for the serial interface.

MICRO-COMPUTER DEVELOPMENT BOARDS



MCDB-A



MCDB-B

Ideally suited for teaching purposes, research projects and micro-computer circuit experiments. Whatever your use WYMER MICRO-COMPUTER DEVELOPMENT BOARDS will make your efforts that much easier. These high quality printed circuit boards enable you to develop a multitude of micro-computer applications:

microprocessor controller circuits, data acquisition terminals, high resolution graphics displays, electronic games, personal computers, memory expansions, word processors, software developments, and numerous other specialist applications.

Each Board Features

- High quality double sided p.c.b. layout
- Solder resist finish
- Spaces for TTL/CMOS logic
- Spaces for MPU and peripheral devices
- Ready laid out memory interconnections
- D-type and ID connector spaces
- 100% through plated holes
- Dimensions 20 cm × 15 cm

In Addition:

MCDB-A accepts up to twelve 24-pin SRAM/ROM/EPROM chips with only 3 control pins unconnected.

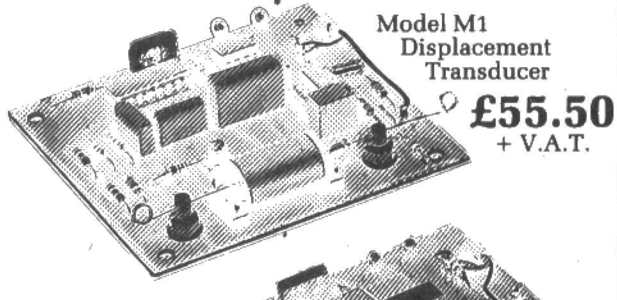
MCDB-B accepts eight 16K, 64K or 256K dynamic RAM chips and two 28/24-pin ROM/EPROM chips.

Price £14.95 inclusive.
Bulk prices available on request.

Available from:
WYMER ELECTRONICS,
8 Briar Place, Dunfermline,
Fife KY11 4BE
Telephone: 0383 418101

WYMER

Pressure & Displacement Transducers for BBC 'B' Micro-Computer



Model M1
Displacement
Transducer
£55.50
+ V.A.T.

Model HP1
Pressure
Transducer
£59.50 + V.A.T.

Available for 0-30, 0-100, 0-160 P.S.I.
Both modules complete with all necessary plugs. No separate power supply or interface necessary.
Supplied against cash with order or official purchase order.

NTC

Details from:
Northern Technical & Chemical Services,
331 East Prescot Road,
Liverpool L14 2DD.
Tel: 051 228 4690 Telex: 667325 Comcab/NTC.G.,

Micro graphics techniques

Mike James continues his series explaining all-you-need-to-know about computer graphics, with a description of two-dimensional sketching techniques.

There are many different approaches to two-dimensional graphics depending on the application in hand. For this reason it is difficult to describe a general theory and method without having a particular application in mind. Three general areas of application that are commonly encountered are:-

- graphs and charts
- cartoon scenes
- technical or exact drawing

Graphs and charts are a specialised area that require a restricted range of graphics techniques to produce. Apart from one or two problems of data handling and scaling, producing graphs and charts is not difficult one you know some general graphics methods and so this application will not be discussed further. Cartoon scenes are drawings or sketches that don't try to represent exactly anything in the real world. For example, a landscape background to a game is a cartoon because, although it may be as convincing as you like, it doesn't attempt to represent any particular landscape. On the other hand technical graphics attempts to display a real object as accurately as possible. The difference between cartoon and technical graphics can also be seen in the way that the information about the shapes involved is entered to the computer. Cartoon graphics generally involves 'drawing' on the screen but technical graphics usually requires exact co-ordinates of the object to be provided. Once the information is in the machine then there is a great deal of overlap between the methods of the two applications. In this month's micrographics the emphasis will be on cartoon graphics and the problems of creating two-dimensional displays from scratch.

Image Storage

There is usually very little point in creating a display if there is no way of storing it for later use or modification. In general micro computers offer three different ways of storing the information necessary to produce a display.

1) Storing the video RAM
As nearly all microcomputer graphics are memory-mapped, one way to store a display is to store the contents of the video RAM. Re-loading the video RAM will immediately restore the display exactly as it was but notice that none of the information about how the display was produced is stored by this method – this can be a disadvantage if you intend modifying it. You also have to store a fixed and large amount of memory no matter what is displayed on the screen. At its most extreme this can involve storing 10K of RAM for an image with one foreground pixel! In practice this is not too much of a problem in that most images worth saving contain rather more detail than one pixel and if storage space is a problem then a data compression algorithm can be used.

2) Program storage
The second way of storing a display is to store the program that produces it! For example, if a program draws a face by a number of calls to subroutines that draw lines and elipses, then loading and running the program will reproduce the display. This form of storage is the most obvious but it has a number of disadvantages. For example, if you want to modify the display then you will have to modify the program. You are also limited to the range of shapes that you have subroutines or commands to produce. The single great advantage of this method is that it doesn't require any special facilities and allows graphics to be mixed in with the rest of a program. In addition it often offers a speed advantage over the other methods.

3) Co-ordinate storage
The third and final way of storing a display is based on storing the co-ordinates of the lines and shapes that make up the display and using a general purpose graphics program to re-draw the display. Separating the data that defines the display from the program that draws it is a very general way of dealing with two-dimensional graphics and has many advantages when it comes to modifying displays.

As an example, consider a display consisting of a single rectangle. The first method would store the display by storing the value at every pixel that made up the screen. The second method would save the program that drew the rectangle using whatever subroutine calls were necessary. The final method would simply store the co-ordinates of each of the four corners of the rectangle.

Thinking about the screen – co-ordinates or pixels

The three different ways of storing an image on the screen given rise to different ways of thinking about it. Storing the image as a sequence of subroutine calls or as a data base of point co-ordinates fosters the attitude that the screen is a collection of co-ordinates. However, storing the screen RAM doesn't involve co-ordinates at all and this encourages a more fundamental view – that the screen is a collection of pixels. This idea is worth exploring a little more and this month's computer graphics looks at ways of changing and influencing pixels directly. This way of working directly with the patterns that the pixels form is a difficult idea at first because we are so used to programming in terms of co-ordinates. The best way of seeing the essence of the approach is to think about what happens when you draw with a more traditional technology – pencil and paper. Unless you are using a drawing board for technical drawing most pencil sketches are not carried out in terms of co-ordinates. You draw a rough line and shade in areas without measuring their exact position. If something doesn't look right then you erase it and modify if. In this sense you are 'moulding' the patterns of dark and light directly. If you change the technology used with this approach to light pen and screen then the same direct interaction with the pixels that form the image is possible – however, surely the computer can offer something more than just the pencil and paper operations of marking and erasing? The answer is most certainly "yes" but you might be surprised at the amount of computer power necessary to achieve it.

Pointing at Points

The underlying data unit of all graphics is the point, or pixel. Obviously if we are going to draw on the screen then there must be some way of entering information about pixels. The only truly co-ordinate-free method of entering information about pixels is the light pen, or similar device, that allows you physically to point at the pixel or pixels that you want to change. Unfortunately, light pens are not as readily available on microcomputers as they should be and so a more familiar co-ordinate based method has to be used. A pixel can be identified by giving its screen co-ordinates but the idea of having to enter long lists of co-ordinates to pick out the pixels that are part of a shape is of course ridiculous. The next most obvious way of indicating a point on the screen is to use a visible 'graphics cursor' that can be moved around the screen by any of a number of methods. The usual way of controlling a graphics cursor is the joystick or (better) a trackball. How these can be used to define a position on the screen was described last month, as was the most primitive – but always available – graphical input device: the keyboard! For the rest of this series the joystick will often be used as the standard input device. Although it is possible to replace the subroutine that reads the joystick with one that reads the keyboard this will miss many of the difficulties and advantages of joystick control described below.

Depositing Pixels – The Brush

Programming a graphics cursor that moves around the screen according to the position of a joystick is relatively easy. The only complication is that to avoid the cursor modifying any image already on the screen it has to be drawn using an 'invert' or 'exclusive or' action. Once you have a graphics cursor under your control you can use it to specify any pixel on the screen – but what next! You could set things up so that pressing a key on the keyboard would change the pixel that the graphics cursor was currently over to a specified colour. Using this idea you could change one pixel at a time – clearly drawing lines and shapes is going to take a long time! Looking back at more traditional drawing methods suggests that an approach that might be worth trying is to copy the action of a paint brush or a pencil. As a paint brush is drawn across the paper or canvas it leaves a trail of paint behind it. This is easy to implement, when a key, D say, is pressed any pixel that the graphics cursor passes over will be changed to the selected colour.

If you try this out (using the program given later) what you will find that it is very difficult to produce anything but 'spidery' sketches; it is like trying to draw with an extremely fine pencil or brush. The idea is good but we haven't implemented the 'brush' idea with sufficient flexibility. A brush should clearly affect more than one pixel at a time – but how many? The answer is that we need a range of 'brushes' varying in thickness and perhaps even in shape. The characteristics of a brush are defined by the shape that it produces when it is moved around the screen. For example, in **Figure 1** you can see a circular brush, a rectangular brush and a 'flower' brush. If you were to move the graphics cursor to a position and select any one of the three brushes then it would produce the shape shown in Figure 1. Moving the

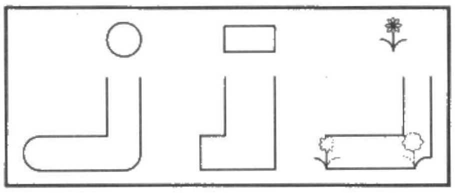


Figure 1. Shape-defined brushes.

graphics cursor with the brush still selected would leave a trail of brush patterns behind it. The circular brush will allow you to draw lines of equal thickness in all directions. The rectangular brush will produce lines of different thickness depending on which way it is moving and the flower brush would normally be used to 'dot' flowers around the screen. Obviously each brush has to be supplied in a range of sizes to enable the thickness of the line to be controlled. Notice also that erasing is done by painting with a brush set to the local background colour.

Given the basic concept of the software generated brush then all sorts of ideas spring to mind for more sophisticated brushes, for example, texture brushes. Instead of changing every point within the brush's area it could affect a pattern of points. A random pattern would give a stippling brush and wavy lines would make drawing a seascape really easy! You can try out using a variable thickness brush and a random stipple by using the program given at the end of this article.

Functional Brushes

All of the brushes described in the last section have one thing in common – they all deposit 'paint' in a way that depends only on their shape and where they are on the screen. A functional brush is once again an obvious extension of the brush idea but it includes the extra facility of looking at what pixels are already on the screen before depositing any 'paint'. For example, suppose that by accident you had 'splashed' some foreground pixels into a region of the screen. You could remove them by taking a very fine brush and touching them out. However, you could use a brush that examines each pixel within its area and changes it to the colour of the majority of its neighbours. By passing such a 'clean up' brush across the splashed area a few times it should be possible to get rid of any mess without damaging what was already present before the splash. Another example of a functional brush is one that will 'sharpen corners'. It can be difficult to draw accurately at the best of times and many corners in an image will be ragged, as shown in **Figure 2.** To sharpen these corners all that is needed is a brush that will introduce the missing pixel where a row and a column of pixels 'cross' as shown in Figure 2. Passing this brush over the image will only change the

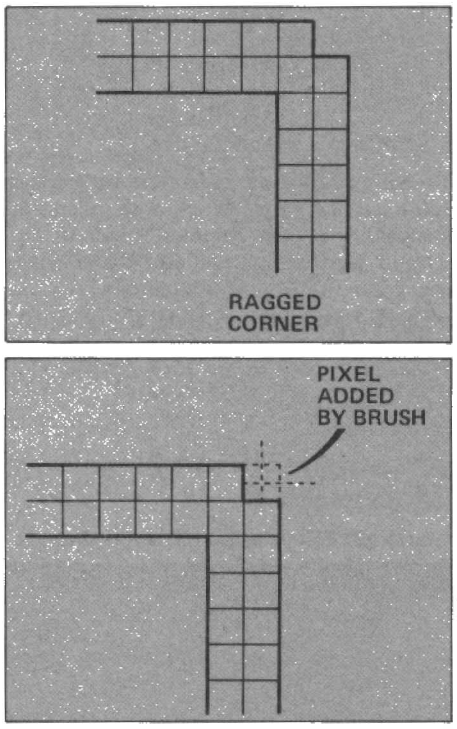


Figure 2. Touching up corners.

points in question. One of the great advantages of a functional brush is that using it you don't run the risk of accidently damaging any part of the image that has already been completed.

As a final example of a functional brush consider the problem of unwanted gaps in lines and curves. A 'fill' brush would enable lines to be patched up without any fear of thickening or distortion. In practice a good fill brush is very difficult to program. Although it is obvious to your eye when a gap should be filled in and when it shouldn't you are using some very sophisticated decision methods. For example, in **Figure 3** there are two equally sized gaps but it is clear that only one of them should be filled. The simplest fill

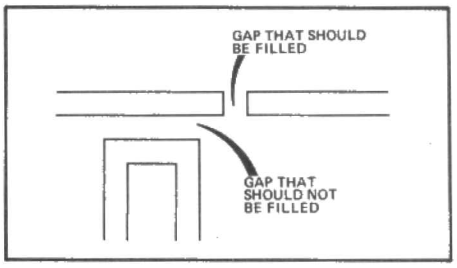


Figure 3. The fill brush.

brush checks the pixels that fall on the perimeter of its shape and, if two of them are in the same colour it will connect them with a straight line. Thus if such a fill brush was passed over the correct gap in figure 3 then it would be filled in. However, if such a brush also happened to include the incorrect gap in its area it too would be filled – so the user beware!

Moving Brushes

Functional brushes will change pixels depending on what is already on the screen, but there are occasions when what is on the screen is correct in form but not in place. The obvious solution is to use the graphics cursor to move the object to a new position. This sounds easy but how do you define the pixels that are to be moved – the 'object' may be clearly visible to you but it isn't to the computer. One answer is to use a fine brush to draw a line around the object and then use the graphics cursor to 'tow' this bag full of pixels across the screen! In practice this is a useful brush but difficult to implement because of the possibly very large number of pixels that have to be moved.

Another form of pixel movement is produced by the 'gravity' brush. This defines an area of 'attraction' that pixels respond to by moving toward the brush. The gravity brush can be used to alter shapes by pulling them in particular directions. Once again the implementation of this sort of brush is very difficult because of the number of pixels that have to be examined and moved.

"the underlying data unit of all graphics is the point, or pixel"

Trajectory Gaps

A problem that has been ignored up to this point is the presence of gaps in the trajectory that a brush follows. This is due to the simple fact that a joystick or a light pen can be moved further than one pixel in the time it takes to sample its position. The effect that this produces on the screen depends on the size of the brush in use but it can result in dotted lines of lines of varying thickness. To a certain extent trajectory gaps are a desirable feature of using brushes because they do give extra control over what the brush produces on the screen. On the other hand the response of a program may be so slow that it is difficult to draw continuous lines without practice at moving the graphics cursor very slowly.

There are two possible approaches to closing the gaps in the trajectory. The first is to simply connect the separate points on the trajectory by straight lines along which the brush moves. This is called linear interpolation and does indeed achieve its objective of removing undesirable gaps. The only trouble is that it will not allow any gaps no matter how fast you move the graphics cursor. The second approach is to allow the brush to drift in the general direction of the graphics cursor. Its speed of drift can be made to depend on how far away from the current position of the cursor the brush is, and so if you want to introduce gaps all that is necessary is to move the graphics cursor very quickly. There is a secondary payoff in making the brush drift towards the graphics cursor and that is the shape of the path that it follows. If the brush's position is updated using –

```
XB=XB+(XG−XB)/N and
YB=YB+(YG−YB)/N
```

where XB,YB is the brush's position and XG,YG is the graphics cursor's position then the path that the brush follows is a parabola. The curvature of the parabola is governed by the value of N, as is the speed with which the brush catches up with the graphics cursor. You might think that having the brush follow a parabolic path to the graphics cursor would be a problem, but in practice as long as the curvature of the parabola is kept small it makes it easier to draw smooth curves!

A Sketch Pad Program

After all this theory about brushes the time has come to give a practical example. The only trouble is that BASIC is not the best language to write a sketch pad program in – it is just too slow for the demands that brushes, functional brushes and movement brushes make on it. As a result it is only possible to demonstrate some of the simpler ideas. The program given below for the BBC Micro includes the following commands:

D	– draw on/off
F	– select foreground paint
B	– select background paint
T	– select thick brush
S	– make thick brush smaller
L	– make thick brush larger
J	– join lines (ie. fill in gaps)
R	– random brush (ie. stipple)
I	– make brush drift to graphics cursor

Most of these commands are ON/OFF type commands. That is pressing the letter once turns the feature on and pressing it a second time turns it off. Drawing with the program is quite simple. The graphics cursor can be moved around the screen using one of the joysticks. Pressing D will turn on the currently selected brush which will deposit pixels in the currently selected colour. For simplicity, only two colours are used called 'background' and 'foreground'. If no brush is selected then the default single pixel – wide fine brush is used to draw. The thick brush can be selected using T and its size can be changed by pressing L for larger and S for smaller. The brush is square simply because any other shape would take too much time to compute in BASIC. A stipple texture brush is available by pressing R (for Random) this will place random blobs of colour within the area selected for the thick brush. The only functional brush that is implemented is a gap filling brush (turned on/off by using J for Join). This brush will connect any two points on the perimeter of the square defined by the current size of the thick brush. So by changing the size of the thick brush you can alter the size of gap that will be filled.

Although the program is written for the BBC Micro it is easy to convert it to run on any other machine that has a joystick because none of the special features of the BBC Micro have been used. The subroutine structure is –

lines	**action**
10-90	main program loop
1000	initialisation
2000	read current position of joystick into X,Y
3000	draw graphics cursor and XG,YG
4000	read keyboard for commands
5000	implement brushes
5100	square brush
5200	stipple brush
5300	gap filling brush
6000	make thick brush smaller
6100	make thick brush bigger
8000	draw vertical line
8100	draw horizontal line

The only other points of interest are that line 3000 sets the graphics action to invert and line 5000 sets the graphics colour to DC (DC=1 gives foreground and DC=0 gives background). The instruction pair

```
MOVE x1,y1
DRAW x2,y2
```

draws a line from x1,y1 to x2,y2

```
PLOT 69,x,y
```

plots a single point at x,y and

```
POINT(x,y)
```

returns the colour value of the pixel at x,y.

BBC sketchpad program

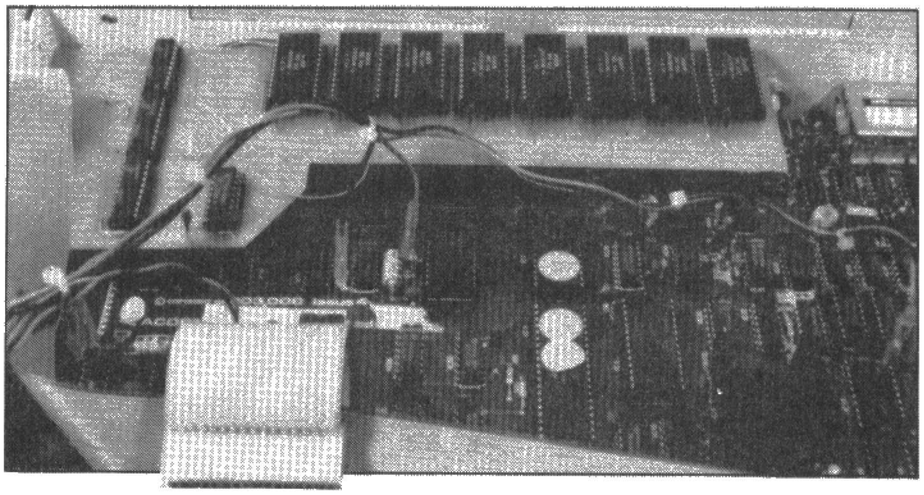
```
  10 MODE 4
  20 GOSUB 1000
  25 GOSUB 3000
  30 GOSUB 2000
  40 GOSUB 3000
  50 XG=X:YG=Y
  60 GOSUB 4000
  70 GOSUB 3000
  90 GOTO 30

1000 XG=100
1010 YG=100
1020 D=-1
1030 DC=1
1040 T=-1
1050 W=16
1060 H=16
1070 C=-1
1080 F=-1
1085 R=-1
1090 G=-1
1100 IT=-1
1110 RETURN

2000 X=INT(1280-ADVAL(1)/51)
2010 Y=INT(ADVAL(2)/64)
2020 RETURN

3000 GCOL 4,0
3010 MOVE XG,YG+8
3020 DRAW XG,YG-8
3030 MOVE XG+8,YG
3040 DRAW XG-8,YG
3050 RETURN

4000 A$=INKEY$(0)
4010 IF A$="F" THEN DC=1
4020 IF A$="B" THEN DC=0
4030 IF A$="D" THEN D=-1*D
4040 IF A$="T" THEN T=-1*T
4050 IF A$="S" THEN GOSUB 6000
4060 IF A$="L" THEN GOSUB 6100
4080 IF A$="J" THEN F=-1*F
4100 IF A$="I" THEN IT=-1*IT
4110 IF A$="R" THEN R=-1*R
4500 IF D=1 THEN GOSUB 5000
4510 RETURN

5000 GCOL 0,DC
5004 IF IT=-1 THEN XB=XG:YB=YG:
     ELSE XB=XB+(XG-XB)/5:YB=
     YB+(YG-YB)/5
5005 IF T=1 THEN GOTO 5100
5006 IF R=1 THEN GOTO 5200
5007 IF F=1 THEN GOTO 5300

5010 PLOT 69,XB,YB
5020 RETURN

5100 REM SQUARE BRUSH
5120 FOR X1=XB-W TO XB+W STEP 4
5130 MOVE X1,YB-H
5140 DRAW X1,YB+H
5150 NEXT X1
5160 RETURN

5200 REM STIPPPLE BRUSH
5210 FOR I=1 TO 2
5220 PLOT 69,XB-W+RND(2*W)
     ,YB-H+RND(2*H)
5230 NEXT I
5240 RETURN

5300 REM FILL BRUSH
5310 FOR Y1=YG-H TO YG+H STEP 4
5320 IF POINT(XG-W,Y1)=1 AND
     POINT(XG+W,Y1)=1 THEN
     GOSUB 8000
5330 NEXT Y1
5340 FOR X1=XG-W TO XG+W STEP 4
5350 IF POINT(X1,YG-H)=1 AND
     POINT(X1,YG+H) THEN
     GOSUB 8100

5360 NEXT X1
5370 RETURN

6000 IF W>4 THEN W=INT(W-4)
6010 IF H>4 THEN H=INT(H-4)
6020 RETURN
6100 W=W+4
6110 H=H+4
6120 RETURN

8000 MOVE XG-W,Y1
8010 DRAW XG+W,Y1
8020 RETURN
8100 MOVE X1,YG-H
8110 DRAW X1,YG+H
8120 RETURN
```

Bring Back Co-ordinates!

With a little practice the sketch pad program can be used to produce reasonable results but drawing with a joystick still isn't easy. Part of the solution lies in the use of some co-ordinate based techniques and these will be described next month. Until then you might like to improve the sketch pad program by adding a routine to save and restore the screen on tape or disc and try experimenting with other brushes. You will, however, probably need more than BASIC to implement anything useful.

Next Month –
Two-dimensional graphics with co-ordinates.

Last month Peter Simpson and Brian Alderwick described the construction and hardware details of the BBC sideways RAM board project. In Part 2, the authors explain the software used in loading and saving a file to the RAM card.



SIDEWAYS RAM BOARD FOR THE BBC MICRO

All the information necessary to use the RAM card was given last month, but for those who are interested a detailed description of the software follows.

The program has been designed to be as tolerant as possible of the condition in which it finds the computer at execution time. This has necessarily made the program more complex, but much reduces the risk of accidentally locking the computer into a state from which the only exit is via the mains on/off switch! This extra protection will be of principal use during the initial phase of entering the program and removing typing errors. It is assumed during execution of the program that the user page zero locations from &70 to &8F do not hold vital information since these are overwritten, that no other code is attached via the *FX247 command and that no extra code is attached

```
  10 REM*******************************
  20 REM*                             *
  30 REM*  RAM CARD INITIALISATION    *
  40 REM*          PROGRAM            *
  50 REM*                             *
  60 REM*            by               *
  70 REM*                             *
  80 REM*       P.W.G. SIMPSON        *
  90 REM*                             *
 100 REM*           and               *
 110 REM*                             *
 120 REM*       B.V. ALDERWICK        *
 130 REM*                             *
 140 REM*         (C) 1983            *
 150 REM*                             *
 160 REM*******************************
 170
 180 ON ERROR GOTO 6790
 190
 200 REM INITIALISE VARIABLES
 210
 220 romselect=&FE30
 230 osfile=&FFDD
 240 oswrch=&FFEE
 250 osbyte=&FFF4
 260 resetvec=&FFFC
 270 romselect=&FE30
 280 temp=&70
 290 loc=&70
 300 locl=&70
 310 loch=&71
 320 from=&70
 330 froml=&70
 340 fromh=&71
 350 to=&72
 360 tol=&72
 370 toh=&73
 380 pb=&7E
 390 newpage=&80
 400 switchstore=&84
 410 clivecl=&208
 420 clivech=&209
 430
 440 PROCmovecode
 450 MODE 7
 460 FOR I%=1 TO 2
 470 VDU141,131:PRINT TAB(7)
     "RAM CARD INITIALISATION"
 480 NEXT
 490 PRINT TAB(9);STRING$(23,"_")
 500 PROCramtestcode
 510 X%=16
 520 REPEAT
 530 X%=X%-1
 540 ramlength=USR(ramtest) AND &FF
 550 UNTIL (X%=0) OR (ramlength<>0)
 560 ramslot=X%
 570 IF ramlength=0 THEN PRINT
     "RAM card is not fitted!":END
 580 PRINT''TAB(11)"RAM is in slot ";
     ramslot
 590 PRINTTAB(11)"RAM present =  ";
     ramlength/4;" K"'''
 600 VDU28,0,24,39,VPOS
 610 CLS
 620 PRINT"  Where do you want the code
     located ?"'
 630 VDU134:PRINT
     "1..Cassette buffer area (&900)"
 640 VDU134:PRINT
     "2..User defined characters area
     (&C00)"
 650 VDU134:PRINT
     "3..Above paged ROM workspace (&";
     ~PAGE-256;")"'
 660 VDU133:PRINT"Option 3 ONLY, new
     PAGE will be &";~PAGE'
 670 *FX15,0
 680 INPUT TAB(8)
     "Enter option number ",L%
 690 IF L%<1 OR L%>3 THEN 610
 700 IF L%=1 THEN location=&900:GOTO0780
 710 IF L%=2 THEN location=&C00:GOTO0780
 720 location=PAGE-256
 730 REM
 740 REM ASSEMBLE CODE THEN DO
 750 REM *FX247, 8, 9 TO SET UP BREAK
 760 REM KEY ACTION THEN RESET.
 770 REM
 780 PROCassemble_lr_sr_code
 790 A%=249
 800 X%=init DIV 256
 810 Y%=0
 820 CALL osbyte
 830 *FX248,0
 840 *FX247,76
 850 CALL reset
 860 END

 870
 880 DEFPROCramtestcode
 890 REM
 900 REM ASSEMBLE CODE TO TEST FOR
 910 REM PRESENCE OF RAM IN SLOT X
 920 REM AND ALSO DETERMINE LENGTH
 930 REM
 940 DIM rtestcode 200
 950 FOR I%=0 TO 2 STEP 2
 960 P%=rtestcode
 970 [          OPT I%
 980 \
 990 \ SAVE PRESENT ROM SLOT NUMBER
1000 \
1010 .ramtest  LDA &F4
1020           PHA
1030 \
1040 \ SELECT ROM X
1050 \
1060           TXA
1070           STA &F4
1080           STA romselect
1090 \
1100 \ SET UP ADDRESSES
1110 \
1120           LDA #&80
1130           STA loch
1140           LDA #0
1150           STA locl
1160           LDY #0
1170           LDX #0
1180 \
1190 \ TEST IF LOCATION &8000 IS RAM
1200 \ IF NOT GO TO rt3
1210 \
1220           JSR ram
1230           BNE rt3
1240           LDX #&40
1250           LDA #0
1260 \
1270 \ SET FIRST BYTE OF EACH 256 BYTE
1280 \ PAGE TO ZERO
1290 \
1300 .rt1      STA (loc),Y
1310           INC loch
1320           DEX
1330           BNE rt1
1340           LDA #&80
1350           STA loch
1360 \
1370 \ TEST FIRST BYTE OF EACH PAGE
1380 \ TO SEE WHETHER IT IS RAM
1390 \ IF NOT GO TO rt3

1400 \
1410 .rt2      JSR ram
1420           BNE rt3
1430           INX
1440           CPX #&40
1450           BEQ rt3
1460           INC loch
1470 \
1480 \ TEST TO SEE WHETHER THIS IS A
1490 \ REFLECTION OF A 1K RAM
1500 \ IF NOT GO TO rt2
1510 \
1520           LDA (loc),Y
1530           BEQ rt2
1540 \
1550 \ SELECT ORIGINAL ROM SLOT
1560 \ AND RETURN
1570 \
1580 .rt3      PLA
1590           STA &F4
1600           STA romselect
1610           TXA
1620           RTS
1630
1640 \
1650 \ TEST IF (loc),Y IS RAM
1660 \ IF SO THEN RETURN WITH EQUAL
1670 \ FLAG SET. IF NOT THEN UNSET
1680 \
1690 .ram      LDA #0
1700           STA (loc),Y
1710           LDA (loc),Y
1720           BNE r1
1730           LDA #&FF
1740           STA (loc),Y
1750           LDA (loc),Y
1760           CMP #&FF
1770 .r1       RTS
1780
1790 ]
1800 NEXT
1810 ENDPROC
1820
1830 DEFPROCmovecode
1840 REM
1850 REM TEST WHETHER THE *FX247,76
1860 REM COMMAND HAS ALREADY BEEN
1870 REM ISSUED.
1880 REM
1890 REM IF ISSUED THEN UNSET IT.
1900 REM
1910 REM CHECK IF PROGRAM IS IN THE
1920 REM CORRECT PLACE IN MEMORY.
1930 REM
1940 REM IF SO THEN RETURN.
1950 REM
1960 REM IF NOT THEN RESET COMPUTER,
1970 REM MOVE THE PROGRAM AND RERUN IT
1980 REM
1990 FORI%=0 TO 2 STEP 2
2000 P%=&C00
2010 [          OPT I%
2020 \
2030 \ TEST WHETHER *FX247 COMMAND
2040 \ HAS BEEN ISSUED.
2050 \
2060 .dis      LDA #247
2070           LDX #0
2080           LDY #&FF
2090           JSR osbyte
2100           CPX #76
2110           BNE d1
2120 \
2130 \ COMMAND HAS BEEN ISSUED SO
2140 \ CANCEL IT.
2150 \
2160           LDX #0
2170           LDY #0
2180           JSR osbyte
2190           JMP d2
2200 \
2210 \ CHECK WHETHER PAGE HAS THE
2220 \ CORRECT VALUE.
2230 \
2240 \ IF SO THEN RETURN.
2250 \
2260 \ IF NOT THEN RESET COMPUTER.
2270 \
2280 .d1       LDA #179
2290           LDX #0
2300           LDY #&FF
```

to any of the operating system vectors. These restrictions are lifted after installation of the new *LR and *SR command code, except that several of the page zero locations will be overwritten during execution of the two commands.

A relatively unpublicised feature of the BBC computer is employed in the program. This is the *FX247 command. In conjuction with the *FX248 and *FX249 commands it is able to set up a machine code jump instruction JMP &XXYY. The JMP op-code is 76 and is supplied by the *FX247,76 command. The address &XXYY is supplied by the *FX248 and *FX249 commands which supply the low and high bytes respectively. Every time that the BREAK key is pressed, including CONTROL/BREAK, the operating system checks to see whether the *FX247,76 command has been issued. If it has not been issued then the standard sequence of events is continued, however, if it has been issued the operating system will jump to the address &XXYY and start to execute the code there. Control can be returned to the standard BREAK sequence of events by executing an RTS instruction at the end of the extra code. This code will be executed every time the break key is pressed, until a *FX247 command is issued whose first parameter is not 76. There is one complication with this technique, in that the code is called not once but twice during the BREAK sequence of events. The first time it is called is with the carry flag clear and before the paged ROMs are interrogated, whilst the second time is with the carry flag set and after the paged ROMs have been interrogated. If you require your code to be executed only once, then you must decide which of the two passes you wish to use and use the carry flag to test for this pass. This technique is employed in subroutine <init>, which is the code executed during the BREAK sequence of events, to allow execution only on the second pass.

"a relatively unpublicised feature of the BBC computer is employed in the program".

In the initialisation program the *FX247 feature is used to continually reconnect the <start> code, which tests for *LR and *SR, to the command line interpreter (CLI). Thus every time the BREAK key is pressed and the command line interpreter vector 'clivec' is returned to its default value, i.e. not pointing at the <start> code, the <init> code will be executed (because *FX247, 8 and 9 point at it) and will change the contents back to point at the <start> code! This is achieved by taking the contents of 'clivec', saving it at 'osclivec' and changing 'clivec' to point at the <start> code. One extra action will be performed if option three has been selected and that is to increment the OSHWM by &100, to make the MOS believe that the paged ROMs have taken 256 bytes of memory more than they actually have. It is these 256 bytes that contain the RAM card code and which are thus protected from the language ROMs such as BASIC.

When a command is passed to the CLI it will first vector through 'clivec' to the <start> code. If the command is *LR or *SR then it will be processed in this code. If it is not either of these then it will be passed onto the CLI by vectoring through 'osclivec' which contains the original contents of 'clivec' and is the address of the MOS CLI routine.

The option to store the operational code above the paged ROM workspace poses a number of problems since PAGE is normally set to this location and the initialisation program will overwrite the beginning of itself when it assembles the extra code. To

```
2310             JSR osbyte
2320             CPX #(PAGE DIV 256)-1
2330             BNE d2
2340             RTS
2350 \
2360 \ SAVE KEYBOARD SWITCH SETTINGS
2370 \ THEN SELECT MODE 7 AS DEFAULT
2380 \ DISPLAY MODE AND RESET COMPUTER
2390 \
2400 .d2         LDA #255
2410             LDX #0
2420             LDY #&FF
2430             JSR osbyte
2440             STX switchstore
2450             LDX #&FF
2460             LDY #0
2470             JSR osbyte
2480             JMP (resetvec)
2490
2500 \
2510 \ SUBROUTINE TO MOVE THIS PROGRAM
2520 \ FROM THE PRESENT VALUE OF PAGE
2530 \ UP OR DOWN IN MEMORY TO THE
2540 \ CORRECT VALUE OF PAGE.
2550 \
2560 .mprog      LDA #0
2570             STA froml
2580             STA tol
2590             STA newpage
2600             STA newpage + 2
2610             STA newpage + 3
2620 \
2630 \ GET FROM HIGH BYTE
2640 \
2650             LDA # PAGE DIV 256
2660             STA fromh
2670 \
2680 \ TEST WHETHER MOVING UP OR
2690 \ DOWN IN MEMORY AND CALL
2700 \ APPROPRIATE ROUTINE.
2710 \
2720             LDA #179
2730             LDX #0
2740             LDY #&FF
2750             JSR osbyte
2760             CPX fromh
2770             BPL mp2
2780 \
2790 \ MOVING DOWN SO MOVE FROM BOTTOM
2800 \ TO TOP
2810 \
2820             INX
2830             STX newpage + 1
2840             STX toh
2850             LDY #0
2860             LDX # (TOP-PAGE) DIV 256
2870             INX
2880 .mp1        LDA (from),Y
2890             STA (to),Y
2900             INY
2910             BNE mp1
2920             INC fromh
2930             INC toh
2940             DEX
2950             BNE mp1
2960             RTS
2970
2980 \
2990 \ MOVING UP SO MOVE FROM TOP TO
3000 \ BOTTOM.
3010 \
3020 .mp2        LDA # TOP DIV 256
3030             STA fromh
3040             INX
3050             STX newpage + 1
3060             TXA
3070             CLC
3080             ADC #(TOP-PAGE) DIV 256
3090             STA toh
3100             LDY #0
3110             LDX #(TOP-PAGE) DIV 256
3120             INX
3130 .mp3        LDA (from),Y
3140             STA (to),Y
3150             INY
3160             BNE mp3
3170             DEC fromh
3180             DEC toh
3190             DEX
3200             BNE mp3
3210             RTS
3220 ]
3230 NEXT
3240 M%=mprog
3250 REM
3260 REM PROGRAM BREAK KEY TO MOVE
3270 REM THIS PROGRAM TO THE NEW
3280 REM VALUE OF PAGE AND RE-RUN IT.
3290 REM
3300 *KEY10"CALL M%|MPAGE=!&80|MOLD|M
     RUN|M"
3310 CALL dis
3320 REM
3330 REM RESTORE KEYBOARD SWITCH
3340 REM SETTINGS ALTERED BY
3350 REM SUBROUTINE dis.
3360 REM
3370 A%=255
3380 X%=?switchstore
3390 Y%=0
3400 CALL osbyte
3410 *KEY10""
3420 ENDPROC
3430
3440 DEFPROCassemble_lr_sr_code
3450 REM
3460 REM ASSEMBLE CODE FOR LR AND
3470 REM SR COMMANDS.
3480 REM
3490 REM ASSEMBLE CODE TO RECONNECT
3500 REM ABOVE CODE TO CLI AFTER A
3510 REM BREAK
3520 REM
3530 FOR I%=0 TO 2 STEP 2
3540 P%=location
3550 [           OPT I%
3560 \
3570 \ CODE TO RECONNECT LR, SR CODE
3580 \ TO THE CLI.
3590 \
3600 \ ONLY EXECUTE THIS CODE WHEN
3610 \ CARRY IS SET.
3620 \
3630 .init       BCC i1
3640             PHP
3650             PHA
3660 \
3670 \ STORE PRESENT CLIVEC CONTENTS
3680 \ IN OSCLIVEC THEN CHANGE CLIVEC
3690 \ TO POINT AT CODE TO TEST FOR
3700 \ LR AND SR.
3710 \
3720             LDA clivecl
3730             STA osclivec
3740             LDA clivech
3750             STA osclivec+1
3760             LDA #start MOD 256
3770             STA clivecl
3780             LDA #start DIV 256
3790             STA clivech
3800 \
3810 \ INSERT EXTRA CODE TO INCREMENT
3820 \ PAGED ROM WORKSPACE POINTERS
3830 \ BUT ONLY IF CODE IS LOCATED
3840 \ ABOVE PAGED ROM WORKSPACE.
3850 \ THEN RETURN.
3860 \
3870 ]
3880
3890 PROCextracode
3900
3910 [           OPT I%
3920             PLA
3930             PLP
3940 .i1         RTS
3950
3960 \
3970 \ ROUTINE TO TEST WHETHER COMMAND
3980 \ LINE CONTAINS "*LR " OR "*SR "
3990 \ STRINGS.

4000 \
4010 .start      STX &F2
4020             STY &F3
4030             LDY #&FF
4040 \
4050 \ IGNORE ANY CHARACTERS WHOSE
4060 \ ASCII CODES ARE LESS THAN 43
4070 \ INCLUDING "*" AND SPACE.
4080 \
4090 .s1         INY
4100             LDA (&F2),Y
4110             CMP # ASC("+")
4120             BMI s1
4130 \
4140 \ TEST FOR "L" AND "S".
4150 \
4160             CMP # ASC("L")
4170             BEQ s3
4180             CMP # ASC("S")
4190             BEQ s3
4200 \
4210 \ FIRST CHARACTER WAS NOT "L"
4220 \ OR "S" HENCE PASS LINE TO
4230 \ CLI.
4240 \
4250 .s2         LDY &F3
4260             JMP (osclivec)
4270
4280 \
4290 \ FIRST CHARACTER WAS EITHER "L"
4300 \ OR "S". STORE IT AT temp THEN
4310 \ TEST WHETHER NEXT CHARACTER IS
4320 \ "R". IF NOT THEN PASS TO CLI.
4330 \
4340 .s3         STA temp
4350             INY
4360             LDA (&F2),Y
4370             CMP # ASC("R")
4380             BNE s2
4390 \
4400 \ CHARACTER WAS "R". NOW TEST
4410 \ WHETHER NEXT CHARACTER IS SPACE.
4420 \ IF NOT THEN PASS TO CLI.
4430 \
4440             INY
4450             LDA (&F2),Y
4460             CMP # ASC(" ")
4470             BNE s2
4480 \
4490 \ LR OR SR COMMAND. GET FIRST
4500 \ CHARACTER TO DETERMINE WHICH
4510 \ THEN GO TO APROPRIATE ROUTINE.
4520 \
4530             LDA temp
4540             CMP # ASC("L")

4550             BNE sram
4560 \
4570 \ LR ROUTINE.
4580 \ INITIALISE VARIABLES, LOAD
4590 \ FILE INTO RAM STARTING AT PAGE,
4600 \ TRANSFER FILE FROM RAM TO
4610 \ RAM CARD AND THEN RESET THE
4620 \ COMPUTER TO MAKE IT ACKNOWLEDGE
4630 \ THE CONTENTS OF THE RAM CARD.
4640 \
4650 .lram       JSR setup
4660             TYA
4670             PHA
4680             LDA #&FF
```

counteract this the program uses PROCmovecode to compare the present value of PAGE with the top of the paged ROM workspace (OSHWM). If PAGE does not equal OSHWM+&100 then the whole BASIC program is moved to OSHWM +&100 and re-run, if it does equal OSHWM +&100 then nothing happens and the procedure ends. The program is moved by initialising an 'intelligent' machine code move routine with the from and to addresses and the length of the program. The BREAK key is then programmed to call the routine, set PAGE to the correct new value (OSHWM + &100), do an OLD and then a RUN. The program will then be in the correct position and the move will not be initiated on the second run. During the move, a *FX247,0 command is executed to switch off any code which might subsequently become corrupted. In a cassette based system the consequence of these actions is that PAGE will be set to &F00, whilst it will be set to &1A00 in a disc based system thus preventing the machine code produced by option three from corrupting the BASIC program producing it. If option three is selected then PAGE will retain the values given above, otherwise it will revert to &E00 and &1900 for the cassette and disc based systems respectively.

There are considerable similarities between the operation of the *LR and *SR routines in particular use of the <setup>, <move> and <file> subroutines. The *LR command loads the specified file from the current filing system starting at OSHWM using the A = 0 osfile call (user guide pp 454, 456). Subroutine <move> then selects the RAM card by storing its slot number at 'romselect' and copies the file into the RAM card finally reselecting the paged ROM which was previously active (BASIC). A jump is then made to the 'reset' vector which is the equivalent of pushing the BREAK key.

"... leaves the user free to concentrate on the important task of developing his software ..."

The operating system, as part of the BREAK sequence, now scans all the paged ROMs including the RAM card and if the contents of the RAM card follow the paged ROM protocol it will be accepted and will behave as any normal paged ROM. Similarly the *SR command first uses subroutine <move>, to copy the contents of the RAM card into main RAM starting at OSHWM. This area of memory is then saved to the current filing system by the A = &FF osfile call and a jump is made to the 'reset' vector to tidy up the program area. Note that *LR and *SR, like the disc commands *BACKUP and *COPY, should not be used from within a BASIC program because they overwrite the area where the program is stored and will consequently corrupt it.

Conclusion

It is hoped that this RAM board will provide an easy way to develop paged ROM software before it is committed to EPROM. The results of changes to such software will be almost immediately available without need to erase and reprogram EPROMs. The code required to operate the board will remain available, once loaded, until the computer is switched off, leaving the user free to concentrate on the important task of developing his software. Now get programming!

The authors are prepared to supply copies of the software, if required for five pounds including postage and packing. Please send orders to the address below stating whether you require cassette, 40 track or 80 track disc.

P. W. G. Simpson,
198, Church Drive, Quedgeley,
GLOUCESTER GL2 6US.

E&CM

```
4690            JSR file
4700            LDY #&80
4710            PLA
4720            JSR move
4730 .reset     JMP (resetvec)
4740
4750 \
4760 \ SR ROUTINE.
4770 \ INITIALISE VARIABLES, SET UP
4780 \ PARAMETER BLOCK FOR OSFILE,
4790 \ MOVE CONTENTS OF RAM CARD INTO
4800 \ RAM STARTING AT PAGE, SAVE RAM
4810 \ TO FILING SYSTEM AND RESET
4820 \ COMPUTER.
4830 \
4840 .sram      JSR setup
4850            STY pb+7
4860            STY pb+11
4870            TYA
4880            CLC
4890            ADC #ramlength
4900            STA pb+15
4910            LDA #&80
4920            JSR move
4930            LDA #0
4940            JSR file
4950            JMP (resetvec)
4960
4970 \
4980 \ SUBROUTINE TO MOVE A BLOCK
4990 \ OF MEMORY FROM ONE PAGE
5000 \ BOUNDARY TO ANOTHER.
5010 \
5020 \ ON ENTRY THE ACCUMULATOR SHOULD
5030 \ CONTAIN THE 'FROM' PAGE AND THE
5040 \ Y REGISTER SHOULD CONTAIN THE
5050 \ 'TO' PAGE
5060 \
5070 \ THE LENGTH OF MEMORY MOVED IS
5080 \ THE LENGTH OF RAM IN THE RAM
5090 \ CARD.
5100 \
5110 .move      STA fromh
5120            STY toh
5130 \
5140 \ GET PRESENT ROM SLOT NUMBER AND
5150 \ SAVE IT, THEN SELECT RAM CARD.
5160 \
5170            LDA &F4
5180            PHA
5190            LDA #ramslot
5200            STA &F4
5210            STA romselect
5220 \
5230 \ INITIALISE VARIABLES.
5240 \
5250            LDX #ramlength
5260            LDY #0
5270            STY froml
5280            STY tol
5290 \
5300 \ MOVE MEMORY.
5310 \
5320 .ml        LDA (from),Y
5330            STA (to),Y
5340            INY
5350            BNE ml
5360            INC fromh
5370            INC toh
5380            DEX
5390            BNE ml
5400 \
5410 \ MEMORY IS MOVED GET SLOT NUMBER
5420 \ OF PREVOUS ROM, RESELECT IT AND
5430 \ RETURN.
5440 \
5450            PLA
5460            STA &F4
5470            STA romselect
5480            RTS
5490
5500 \
5510 \ SUBROUTINE TO INITIALISE
5520 \ VARIABLES COMMON TO LR AND SR.
5530 \
5540 \ ON ENTRY Y SHOULD CONTAIN THE
5550 \ OFFSET TO (&F2) WHICH TOGETHER
5560 \ POINT AT THE START OF THE FILE
5570 \ NAME.
5580 \
5590 \ ON EXIT Y CONTAINS THE PAGE
5600 \ NUMBER OF THE TOP OF PAGED ROM
5610 \ WORKSPACE.
5620 \
5630 \ FIRST CLEAR OSFILE PARAMETER
5640 \ BLOCK BY SETTING UP FOUR DOUBLE
5650 \ WORDS OF &FFFF0000
5660 \
5670 .setup     LDX #18
5680 .sul       DEX
5690            LDA #&FF
5700            STA pb,X
5710            DEX
5720            STA pb,X
5730            LDA #0
5740            DEX
5750            STA pb,X
5760            DEX
5770            STA pb,X
5780            CPX #2
5790            BNE sul
5800 \
5810 \ CALCULATE ADDRESS OF START OF
5820 \ FILE NAME AND STORE IT IN
5830 \ PARAMETER BLOCK
5840 \
5850            CLC
5860            TYA
5870            ADC &F2
5880            STA pb
5890            LDA #0
5900            ADC &F3
5910            STA pb+1
5920 \
5930 \ SELECT MODE 7 SO THAT A 16K
5940 \ FILE WILL NOT OVERWRITE THE
5950 \ SCREEN .
5960 \
5970            LDA #22
5980            JSR oswrch
5990            LDA #7
6000            JSR oswrch
6010 \
6020 \ GET THE ADDRESS OF THE TOP OF
6030 \ THE PAGED ROM WORKSPACE AND
6040 \ STORE IN PARAMETER BLOCK.
6050 \
6060            LDA #131
6070            JSR osbyte
6080            STY pb+3
6090            RTS
6100
6110 \ SUBROUTINE TO LOAD AND SAVE
6120 \ FILES.
6130 \
6140 \ SET UP X AND Y REGISTERS THEN
6150 \ CALL OSFILE AND RETURN.
6160 \
6170 .file      LDX #pb MOD 256
6180            LDY #pb DIV 256
6190            JSR osfile
6200            RTS
6210
6220 \ RESERVE TWO BYTES FOR THE
6230 \ ORIGINAL CONTENTS OF THE CLI
6240 \ VECTOR.
6250 \
6260 .osclivec BRK
6270            BRK
6280
6290 ]
6300 NEXT
6310 ENDPROC
6320
6330 DEFPROCextracode
6340 REM
6350 REM PROCEDURE  WHICH WILL
6360 REM ASSEMBLE EXTRA CODE NEEDED
6370 REM FOR OPTION THREE.
6380 REM
6390 REM ASSEMBLY ONLY TAKES PLACE
6400 REM WHEN OPTION THREE IS SELECTED
6410 REM
6420 IF location<>(PAGE-256) THEN
     ENDPROC
6430 [          OPT I%
6440 \
6450 \ GET PRESENT VALUE OF THE TOP OF
6460 \ PAGED ROM WORKSPACE.
6470 \
6480            LDA #179
6490            LDX #0
6500            LDY #&FF
6510            JSR osbyte
6520 \
6530 \ INCREMENT THIS VALUE.
6540 \
6550            INX
6560            TXA
6570            PHA
6580 \
6590 \ STORE NEW VALUE IN TWO
6600 \ OPERATING SYSTEM VARIABLES.
6610 \
6620            LDY #0
6630            LDA #179
6640            JSR osbyte
6650            PLA
6660            TAX
6670            LDY #0
6680            LDA #180
6690            JSR osbyte
6700 ]
6710 ENDPROC
6720
6730 REM********************************
6740 REM                               *
6750 REM   ERROR HANDLING ROUTINE      *
6760 REM                               *
6770 REM********************************
6780
6790 ON ERROR OFF
6800 VDU26,31,0,24
6810 PRINT"OH DEAR, MASTER, I AM SORRY
     TO HAVE TO"'"INFORM YOU THAT :-"
6820 REPORT:PRINT" HAS OCCURED AT
     LINE ";ERL
6830 PRINT'"PAGE IS CURRENTLY SET TO &"
     ;~PAGE
6840 END
>
```

A cross-assembler for the MC68705s

To make the most of the power offered by the Motorola MC68705 one-chip microcontroller, an assembler program is absolutely vital. Richard Whitlock introduces an assembler routine which can run on any home computer using BASIC.

These tiny systems have twice or four times the program memory space of the KIM-1s and Acorn System 1s on which many people started machine code programming, five or six years ago, when there was no other sort of programming possible on machines costing less than £500. In those days "hand-assembly" was taken for granted as two and three hundred byte programs were ground out, taking perhaps all the spare time in a week or more, and resulting in a real sense of achievement at the end of that time. Now there may be a few masochists about, who would get a buzz out of "hand-assembling" a full-blown 3.8 kbyte 68705R3 or 'U3 application program, but I suspect that they are a vanishing few. For the rest of you here is a 68705 Cross-Assembler that should run with only slight modification on any home computer with BASIC.

In the interests of portability the program is written in a subset of Microsoft 8K BASIC – as near an international standard as one is likely to find. All arithmetic operations are integer only, and only a few of the more commonly implemented BASIC functions are employed. While little difficulty should be experienced simply translating the program into the dialect of BASIC native to a particular machine, the owners of machines supporting the more sophisticated dialects may find that considerable savings in the amount of memory tied up by the assembler routines can be made by more elaborate rewriting.

What The BASIC Cross-Assembler Does

In essence the program presented in **Listing 1** is intended to form the latter part of a run-time package that also incorporates the Assembly Language statements to be assembled. It is intended that the missing lines (1-10,000) should hold the Assembly Language statements, each embedded in a BASIC DATA statement. The justification for this messy form of operand entry is that it greatly simplifies memory management on the one hand, and contributes to the security with which the Assembly Language statements are stored and manipulated on the other. Apart from the work space normally appropriated by the BASIC Interpreter, this Assembler uses only a two or four kilobyte buffer, to receive the assembled code, ready to be transferred to the EPROM from which the 68705 will program itself; two arrays which are filled only at run-time; and the actual lines of the program itself. During the

TABLE 7 – M6805 HMOS FAMILY OPCODE MAP

	Bit Manipulation		Branch	Read/Modify/Write					Control		Register/Memory						
	BTB	BSC	REL	DIR	INH	INH	IX1	IX	INH	INH	IMM	DIR	EXT	IX2	IX1	IX	
Low \ Hi	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	Hi \ Low
0 0000	10 BRSET0 3 BTB	7 BSET0 2 BSC	4 BRA 2 REL	6 NEG 2 DIR	4 NEG 1 INH	4 NEG 1 INH	7 NEG 2 IX1	6 NEG 1 IX	9 RTI 1 INH		2 SUB 2 IMM	4 SUB 2 DIR	5 SUB 3 EXT	6 SUB 3 IX2	5 SUB 2 IX1	4 SUB 1 IX	0 0000
1 0001	10 BRCLR0 3 BTB	7 BCLR0 2 BSC	4 BRN 2 REL						6 RTS 1 INH		2 CMP 2 IMM	4 CMP 2 DIR	5 CMP 3 EXT	6 CMP 3 IX2	5 CMP 2 IX1	4 CMP 1 IX	1 0001
2 0010	10 BRSET1 3 BTB	7 BSET1 2 BSC	4 BHI 2 REL								2 SBC 2 IMM	4 SBC 2 DIR	5 SBC 3 EXT	6 SBC 3 IX2	5 SBC 2 IX1	4 SBC 1 IX	2 0010
3 0011	10 BRCLR1 3 BTB	7 BCLR1 2 BSC	4 BLS 2 REL	6 COM 2 DIR	4 COMA 1 INH	4 COMX 1 INH	7 COM 2 IX1	6 COM 1 IX	11 SWI 1 INH		2 CPX 2 IMM	4 CPX 2 DIR	5 CPX 3 EXT	6 CPX 3 IX2	5 CPX 2 IX1	4 CPX 1 IX	3 0011
4 0100	10 BRSET2 3 BTB	7 BSET2 2 BSC	4 BCC 2 REL	6 LSR 2 DIR	4 LSRA 1 INH	4 LSRX 1 INH	7 LSR 2 IX1	6 LSR 1 IX			2 AND 2 IMM	4 AND 2 DIR	5 AND 3 EXT	6 AND 3 IX2	5 AND 2 IX1	4 AND 1 IX	4 0100
5 0101	10 BRCLR2 3 BTB	7 BCLR2 2 BSC	4 BCS 2 REL								2 BIT 2 IMM	4 BIT 2 DIR	5 BIT 3 EXT	6 BIT 3 IX2	5 BIT 2 IX1	4 BIT 1 IX	5 0101
6 0110	10 BRSET3 3 BTB	7 BSET3 2 BSC	4 BNE 2 REL	6 ROR 2 DIR	4 RORA 1 INH	4 RORX 1 INH	7 ROR 2 IX1	6 ROR 1 IX			2 LDA 2 IMM	4 LDA 2 DIR	5 LDA 3 EXT	6 LDA 3 IX2	5 LDA 2 IX1	4 LDA 1 IX	6 0110
7 0111	10 BRCLR3 3 BTB	7 BCLR3 2 BSC	4 BEQ 2 REL	6 ASR 2 DIR	4 ASRA 1 INH	4 ASRX 1 INH	7 ASR 2 IX1	6 ASR 1 IX		2 TAX 1 INH		5 STA 2 DIR	6 STA 3 EXT	7 STA 3 IX2	6 STA 2 IX1	5 STA 1 IX	7 0111
8 1000	10 BRSET4 3 BTB	7 BSET4 2 BSC	4 BHCC 2 REL	6 LSL 2 DIR	4 LSLA 1 INH	4 LSLX 1 INH	7 LSL 2 IX1	6 LSL 1 IX		2 CLC 1 INH	2 EOR 2 IMM	4 EOR 2 DIR	5 EOR 3 EXT	6 EOR 3 IX2	5 EOR 2 IX1	4 EOR 1 IX	8 1000
9 1001	10 BRCLR4 3 BTB	7 BCLR4 2 BSC	4 BHCS 2 REL	6 ROL 2 DIR	4 ROLA 1 INH	4 ROLX 1 INH	7 ROL 2 IX1	6 ROL 1 IX		2 SEC 1 INH	2 ADC 2 IMM	4 ADC 2 DIR	5 ADC 3 EXT	6 ADC 3 IX2	5 ADC 2 IX1	4 ADC 1 IX	9 1001
A 1010	10 BRSET5 3 BTB	7 BSET5 2 BSC	4 BPL 2 REL	6 DEC 2 DIR	4 DECA 1 INH	4 DECX 1 INH	7 DEC 2 IX1	6 DEC 1 IX		2 CLI 1 INH	2 ORA 2 IMM	4 ORA 2 DIR	5 ORA 3 EXT	6 ORA 3 IX2	5 ORA 2 IX1	4 ORA 1 IX	A 1010
B 1011	10 BRCLR5 3 BTB	7 BCLR5 2 BSC	4 BMI 2 REL							2 SEI 1 INH	2 ADD 2 IMM	4 ADD 2 DIR	5 ADD 3 EXT	6 ADD 3 IX2	5 ADD 2 IX1	4 ADD 1 IX	B 1011
C 1100	10 BRSET6 3 BTB	7 BSET6 2 BSC	4 BMC 2 REL	6 INC 2 DIR	4 INCA 1 INH	4 INCX 1 INH	7 INC 2 IX1	6 INC 1 IX		2 RSP 1 INH		3 JMP 2 DIR	4 JMP 3 EXT	5 JMP 3 IX2	4 JMP 2 IX1	3 JMP 1 IX	C 1100
D 1101	10 BRCLR6 3 BTB	7 BCLR6 2 BSC	4 BMS 2 REL	6 TST 2 DIR	4 TSTA 1 INH	4 TSTX 1 INH	7 TST 2 IX1	6 TST 1 IX		2 NOP 1 INH	8 BSR 2 REL	7 JSR 2 DIR	8 JSR 3 EXT	9 JSR 3 IX2	8 JSR 2 IX1	7 JSR 1 IX	D 1101
E 1110	10 BRSET7 3 BTB	7 BSET7 2 BSC	4 BIL 2 REL								2 LDX 2 IMM	4 LDX 2 DIR	5 LDX 3 EXT	6 LDX 3 IX2	5 LDX 2 IX1	4 LDX 1 IX	E 1110
F 1111	10 BRCLR7 3 BTB	7 BCLR7 2 BSC	4 BIH 2 REL	6 CLR 2 DIR	4 CLRA 1 INH	4 CLRX 1 INH	7 CLR 2 IX1	6 CLR 1 IX		2 TXA 1 INH		5 STX 2 DIR	6 STX 3 EXT	7 STX 3 IX2	6 STX 2 IX1	5 STX 1 IX	F 1111

Abbreviations for Address Modes

INH	Inherent
IMM	Immediate
DIR	Direct
EXT	Extended
REL	Relative
BSC	Bit Set/Clear
BTB	Bit Test and Branch
IX	Indexed (No Offset)
IX1	Indexed, 1 Byte (8-Bit) Offset
IX2	Indexed, 2 Byte (16-Bit) Offset

LEGEND

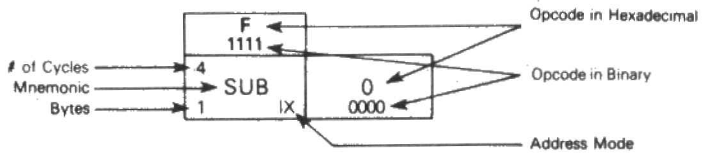


Figure 1. M6805 HMOS family instruction set opcode map. NOTE: To simplify the design of the assembler program, BTB and BSC mnemonics have been abbreviated to three characters, like all other mnemonics. Thus BRSET = BBS: BRCLR = BBC: BSET = BST and BCLR = BCL.

lengthy process of entering the source code (the Assembly Language statements) all the line editing features of the particular machine's BASIC are automatically available without any memory overhead. If it is necessary to break off from a session of entering the source code, a simple program save operation will safely store the present position, without any recourse to complicated block saves. This is no small consideration if, like me, you intend to end your sessions in the small hours of the morning, when well past your best. Conversely when starting up again a single program load restores the former position.

When the source code is all entered and checked, the whole composite program is RUN. A variety of errors can be detected during the first and second passes by the Assembler Routines through the source code. Each time an error is detected the assembly process is aborted, an error message printed to screen, and the program returns to BASIC Command Mode, where the user may LIST and EDIT the offending line. This done the program is RUN again. When all errors are rectified and the program reaches the end of the second pass, the assembled code is in fact then available in the buffer and, if no print-out is required, the process can be stopped there. If, however, a listing is required, then a third pass is necessary to generate the data required. This is because no intermediate data is stored during either the first or second passes, to avoid reducing the amount of memory space available for storing source code. Nothing is printed as the second pass is under way, even though the final information is available at this stage, because there is still the possibility of errors being detected with the corresponding possibility of waste of time and paper if a listing is aborted.

While overall run time will vary widely, depending largely on the number of errors detected, it will normally be possible to assemble a fully entered 1.8 kbyte program in less than two hours. An improvement of some 1000% over the author's best unaided efforts to date.

Syntax of the Assembly Language Statements

While most BASIC dialects are fairly permissive in matters like the number of spaces inserted between the elements of a statement, this cross-assembler, in common with many others, demands careful attention to the spacing of the elements of the source code statements. This is because the source code statements are stored and retrieved as single strings, which are then "sliced up" to recover the information on addressing modes etc. that they contain.

As can be seen from the 68705 Opcode Map (see **Fig 1**), there are six types of instruction grouped by the available addressing modes. **Figs 2-7** show the forms of each instruction type which are accepted by the assembler routines. Note that in source code statements actual hexadecimal values are only used in Immediate mode instructions. In all other addressing modes memory locations are referred to by means of labels. The actual addresses to which these

Figure 2

REGISTER/MEMORY INSTRUCTIONS

Immediate Mode:	SUB@ 6C	
Direct Mode:	SUB LABEL	The value assigned to the label "LABEL" determines whether one or two post-bytes are assembled.
Extended Mode:	SUB LABEL	
Indexed/16-bit Offset:	SUB(LABEL)	
Indexed/8-bit Offset:	SUB(LABEL)	
Indexed/No Offset:	SUB()	

Figure 3

READ/MODIFY/WRITE INSTRUCTIONS

Direct Mode:	NEG LABEL	Labels evaluated to a 16-bit value will cause an Error Message to be shown.
Inherent(Accumulator):	NEGA	
Inherent(Index Register):	NEGX	
Indexed/8-bit Offset:	NEG(LABEL)	
Indexed/No Offset:	NEG()	

Figure 4

INHERENT MODE CONTROL INSTRUCTIONS

Inherent Mode:	RTI

Figure 5

CONVENTIONAL BRANCH INSTRUCTIONS

Relative Mode:	BRA LABEL	Labels evaluated as being outside the allowed branching range will cause an Error Message to be shown.

Figure 6

BIT TEST AND BRANCH INSTRUCTIONS

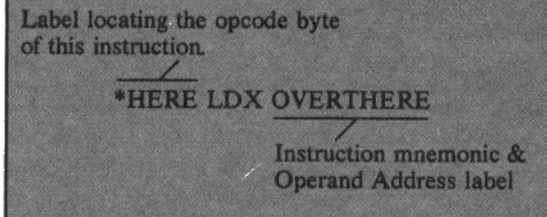
Direct + Relative Mode:	BBS5 LABEL1 LABEL2	LABEL1 must point to RAM or I/O – LABEL2 must evaluate to an address within branching range – or Error shown.
	Bit No. must be in range 0-7 or Error shown.	

Figure 7

BIT SET/CLEAR INSTRUCTIONS

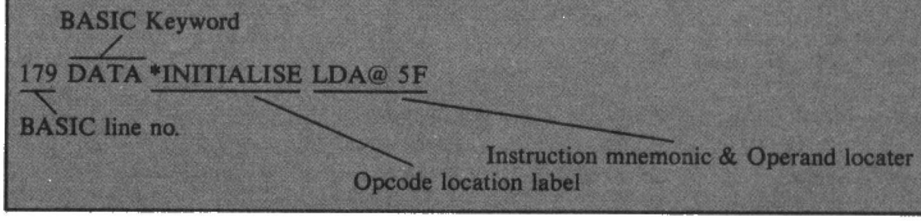
Direct Mode:	BST3 LABEL	LABEL must point to RAM or I/O or Error shown.
	Bit No. must be in range 0-7 or Error shown.	

Figure 8


Label locating the opcode byte of this instruction.
*HERE LDX OVERTHERE
Instruction mnemonic & Operand Address label

N.B. The * is vital; without it the program will try to treat HERE as an Instruction mnemonic.

Figure 9


BASIC Keyword
179 DATA *INITIALISE LDA@ 5F
BASIC line no.
Instruction mnemonic & Operand locater
Opcode location label

labels refer are either pre-defined by means of the EQU assembler pseudo-operation, if they are in page zero below the start of executable code, or are calculated by the assembler itself if they lie within the body of executable code. **Fig 8** shows the addition of a label to a typical source code line.

Figure 10

```
123 DATA *SELF BBS2 PORTA
    SELF

136 DATA        BSTI PORTA
137 DATA        BBC0 PORTA CONT
138 DATA *CONT
                BCL1 PORTA
139 DATA        ASR STORE
```

A whole source code statement, embedded in a BASIC language DATA statement will appear as shown in **Fig 9.** Some caution will have to be exercised in translating this format into a form acceptable to some BASIC dialects. Most BASICs will quite happily accept embedded spaces within a string to be retrieved by a READ operation, rejecting only commas, and in some cases semi-colons, which are used to separate consecutive data items within a single DATA statement, but leading spaces may be a cause for concern. A single space between the DATA keyword and the source code string is, as far as I have been able to ascertain, universally acceptable, but if it is desired to use the LIST command of a particular BASIC to obtain a properly laid out source code listing, (as shown in **Fig 10**), it must be checked that the multiple spaces between the DATA keyword and the source code string, where there is no "front-end" label present, are not attached to the source code string by the READ operation of the same BASIC interpreter. If they are picked up the result will be disastrous!

Note that labels used in the source code to be assembled by this program must not contain any of the disallowed characters dictated by the particular BASIC interpreter used, or embedded spaces and brackets, which are disallowed by the assembler program itself. Other than these restrictions the user is free to make the labels as long and as complex as desired, though do remember that they burn up RAM. The asterisk, as the first character of the label when it is first used as a location marker, is compulsory. The assembler looks for the asterisk as an indication that it has to process a location marker before getting on with assembling the actual machine instruction embodied in the source code line. If the asterisk is not included the assembler will try to match the first three characters of the label name with the list of allowable instruction mnemonics, with unpredictable results.

If the same label name is used, by an oversight, to designate two or more locations in the same program to be assembled, all subsequent references to that label name will be directed to the first address assigned to or calculated for that label name and no error will be detected, unless, that is, the confusion causes a branch to be deemed out of range or a disallowed addressing mode to be called for for a particular class of instruction, eg. extended mode for a READ/MODIFY/WRITE instruction. To control the assembly process a number of instructions that have nothing to do with the instruction set of the micro-controller have to be used to communicate with the assembler routines. These are traditionally termed "pseudo-operations". Commercial assemblers often have a dozen or more of these instructions, this program has just three:

ORG – this instruction has as its operand a four digit hexadecimal number, which is loaded into the BASIC variable pointing to the place in the assembled machine code buffer where the next byte of code will be placed. It is used initially, and as often as necessary subsequently, to move the point to which code is directed and from which relative addressing is calculated. There is no built-in safeguard against overwriting previously assembled code if the ORG operation is used to move the point of assembly back to an area of memory space which has already been filled. See **Fig 11A** for the form of the ORG source code line.

EQU – this instruction is used to assign an address value to a label. It is used primarily to label RAM and I/O locations which lie outside the body of assembled code and thus cannot have their address values determined by calculation. It can also be used to provide the cross-links between two bodies of code to be assembled at different times. The operands of this instruction are the label name and a four digit hexadecimal number which specifies the address value. See **Fig 11 B** for the form of the EQU source code line.

DAT – this instruction is used to insert a byte of any desired value into the stream of assembled code directed to the buffer. It does not determine the address at which the byte will be stored, of itself, but merely places it in the next available location. See **Fig 11 C** for the form of the DAT source code line.

Figure 11

```
A).  456  DATA ORG 0080         — Starts or restarts assembly at 0080H.
B).  379  DATA *LOOPV EQU 0020  — Labels RAM location 0020H as LOOPV.
C).  155  DATA *7SEG DAT 3F     — Inserts byte 3FH in next free location, and
                                  labels that location 7SEG.
```

Operating Notes

1. Memory Space Requirements: The assembler routines of LISTING 1 occupy some 5600 bytes as stored in MBASIC tokenised form. A typical source code line embedded in a DATA statement, including BASIC line number and end of line marker, occupies between 13 and 25 bytes (assuming five-character label names), with an empirically derived average length of about 16 bytes. There is a further memory overhead of 2 + 3 + LEN(LABEL), in the case considered 10 bytes for each label used. Since 1Kbyte of assembled machine code will contain something like 500 instructions, perhaps 100 labels and a few pseudo-operations, it can be seen that something like 22Kbytes of RAM, accessible to BASIC, will be required to hold and assemble a 1.8Kbyte application program for a 68705P3, as a single operation, and a further 2Kbytes of RAM above the area used directly by BASIC to hold the assembled code. The corresponding figures for a 3.8Kbyte 68705R3 or 'U3 application program assembled as a single unit would be 40Kbytes for direct BASIC use and 4Kbytes for the assembled code buffer.

2. Breaking it down into blocks: The above figures are all very encouraging for the owners of 48K RAM systems, or even those with 32K and modest aspirations, but what about those who have only 16K RAM? Using the assumptions outlined above, a 16K system will be able to hold and assemble the source code for about 0.93Kbytes of assembled code, given a 2K buffer allocation. Thus it should be possible to assemble a 1.8Kbyte program in two operations. Practical difficulties that must be overcome include safeguarding the code assembled in the first stage of the process while loading the new BASIC program and keeping track of cross-references to labels in the two parts of the source code text.

3. Safeguarding the assembled code: It is a feature of both the 2732 EPROM and the 2532 EPROM that individual locations may be programmed at random without affecting the future programmability of the remaining locations, also locations that are not required to be programmed at a particular moment may be programmed with FF$_H$ on a number of occasions before a Definitive value is finally programmed into them. Thus, if the buffer that receives the assembled code is initially cleared to all FF$_H$ bytes, the locations not receiving assembled code when the first part of a program is assembled will still contain FF$_H$, and the whole buffer can be 'burned' into the transfer EPROM before the second part of the program is loaded and assembled. A second 'burn' will then fill up the blank part of the EPROM.

LISTING 1 The assembler routines

```
4400 PRINT CHR$(26):PRINT"ASSEMBLER IN BASIC FOR MOTOROLA M68705":PRINT
4410 INPUT"BASE ADDRESS OF ASSEMBLED CODE BUFFER";P%:PRINT
4420 INPUT"HOW MANY LINES OF SOURCE CODE TO BE ASSEMBLED";D%:PRINT
4430 INPUT"HOW MANY LABELS HAVE BEEN USED";C%:PRINT:PRINT
4440 DIM L$(C%),A(C%):LET Z%=1
4450 PRINT"FIRST PASS RUNNING":PRINT:PRINT
4460 LET K%=0:LET Q%=0
4462 FOR X%=1 TO D%
4465 IF Z%=3 THEN LPRINT X%;
4467 GOSUB 17000
4470 GOTO 5000


4500 GOSUB 17150
4505 NEXT X%
4510 LET Z%=Z%+1
4520 IF Z%=4 THEN 4540
4525 IF Z%=3 THEN PRINT"PRINT-OUT":PRINT:PRINT:RESTORE:GOTO 4460
4530 PRINT"SECOND PASS RUNNING":PRINT:PRINT:RESTORE:GOTO 4460
4540 PRINT:PRINT:PRINT"ASSEMBLY COMPLETE":LPRINT:LPRINT:LPRINT"ASSEMBLY COMPLETE
"

              ----------------------------------------------
              :  SPACE FOR AN EPROM PROGRAMMER DRIVER ROUTINE  :
              :  TO DUMP THE ASSEMBLED CODE TO EPROM BEFORE A  :
              :               GREMLIN GETS AT IT!              :
              ----------------------------------------------

4990 END


5000 READ S$:LET P$=S$
5010 IF LEFT$(S$,1)="*"THEN GOSUB 6000
5020 LET M$=LEFT$(S$,3):LET S$=RIGHT$(S$,(LEN(S$)-3))
5030 IF M$="ADC"THEN LET B%=169:GOTO 6200
5040 IF M$="ADD"THEN LET B% =171:GOTO 6200
5050 IF M$="AND"THEN LET B% =164:GOTO 6200
5060 IF M$="ASR"THEN LET B% =55:GOTO 6100
5070 IF M$="BCC"THEN LET O% =36:GOTO 7000
5080 IF M$="BCL"THEN LET B% =17:GOTO 8000
5090 IF M$="BCS"THEN LET O% =37:GOTO 7000
5100 IF M$="BEQ"THEN LET O% =39:GOTO 7000
5110 IF M$="BHC"THEN LET O% =40:GOTO 7000
5120 IF M$="BHF"THEN LET O% =41:GOTO 7000
5130 IF M$="BHI"THEN LET O% =34:GOTO 7000
5140 IF M$="BHS"THEN LET O% =36:GOTO 7000
5150 IF M$="BIH"THEN LET O% =47:GOTO 7000
5160 IF M$="BIL"THEN LET O% =46:GOTO 7000
5170 IF M$="BIT"THEN LET B% =165:GOTO 6200
5180 IF M$="BLO"THEN LET O% =37:GOTO 7000
5190 IF M$="BLS"THEN LET O% =35:GOTO 7000
5200 IF M$="BMC"THEN LET O% =44:GOTO 7000
5210 IF M$="BMI"THEN LET O% =43:GOTO 7000
5220 IF M$="BMS"THEN LET O% =45:GOTO 7000
5230 IF M$="BNE"THEN LET O% =38:GOTO 7000
5240 IF M$="BPL"THEN LET O% =42:GOTO 7000
5250 IF M$="BRA"THEN LET O% =32:GOTO 7000
5260 IF M$="BRN"THEN LET O% =33:GOTO 7000
5270 IF M$="BBC"THEN LET B% =1:GOTO 9000
5280 IF M$="BBS"THEN LET B% =0:GOTO 9000
5290 IF M$="BST"THEN LET B% =16:GOTO 8000
5300 IF M$="BSR"THEN LET O% =173:GOTO 7000
5310 IF M$="CLC"THEN LET O% =152:GOTO 9500
5320 IF M$="CLI"THEN LET O% =154:GOTO 9500
5330 IF M$="CLR"THEN LET B% =63:GOTO 6100
5340 IF M$="CMP"THEN LET B% =161:GOTO 6200
5350 IF M$="COM"THEN LET B% =51:GOTO 6100
5360 IF M$="CPX"THEN LET B% =163:GOTO 6200
5370 IF M$="DAT" THEN 13000
5380 IF M$="DEC"THEN LET B% =58:GOTO 6100
5390 IF M$="EOR"THEN LET B% =168:GOTO 6200
5400 IF M$="EQU" THEN 15000
5410 IF M$="INC"THEN LET B% =60:GOTO 6100
5420 IF M$="JMP"THEN LET B% =188:GOTO 6200
5430 IF M$="JSR"THEN LET B% =189:GOTO 6200
5440 IF M$="LDA"THEN LET B% =166:GOTO 6200
5450 IF M$="LDX"THEN LET B% =174:GOTO 6200
5460 IF M$="LSL"THEN LET B% =56:GOTO 6100
5470 IF M$="LSR"THEN LET B% =52:GOTO 6100
5480 IF M$="NEG"THEN LET B% =48:GOTO 6100
5490 IF M$="NOP"THEN LET O% =157:GOTO 9500
5500 IF M$="ORA"THEN LET B% =170:GOTO 6200
5510 IF M$="ORG"THEN 14000
5520 IF M$="ROL"THEN LET B% =57:GOTO 6100
5530 IF M$="ROR"THEN LET B% =54:GOTO 6100
5540 IF M$="RSP"THEN LET O% =156:GOTO 9500
5550 IF M$="RTI"THEN LET O% =128:GOTO 9500
5560 IF M$="RTS"THEN LET O% =129:GOTO 9500
5570 IF M$="SBC"THEN LET B% =162:GOTO 6200
5580 IF M$="SEC"THEN LET O% =153:GOTO 9500
5590 IF M$="SEI"THEN LET O% =155:GOTO 9500
5600 IF M$="STA"THEN LET B% =183:GOTO 6200
5610 IF M$="STX"THEN LET B% =191:GOTO 6200
5620 IF M$="SUB"THEN LET B% =160:GOTO 6200
5630 IF M$="SWI"THEN LET O% =130:GOTO 9500
5640 IF M$="TAX"THEN LET O% =151:GOTO 9500
5650 IF M$="TST"THEN LET B% =61:GOTO 6100
5660 IF M$="TXA"THEN LET O% =159:GOTO 9500
5670 PRINT"MNEMONIC ERROR IN LINE ";X%:END
```

```
6000 LET L$=MID$(S$,2,1):LET S$=RIGHT$(S$,(LEN(S$)-2))
6010 IF LEFT$(S$,1)=" " THEN 6050
6020 LET L$=L$+LEFT$(S$,1):GOSUB 16000
6030 GOTO 6010
6050 GOSUB 16000:LET K%=K%+1:LET L$(K%)=L$
6060 LET A(K%)=Q%:RETURN


6100 LET T%=1:LET M$=LEFT$(S$,1):GOSUB 16000
6120 IF M$=" "THEN LET M% =0:GOTO 6520
6130 IF M$="A"THEN LET M% =1:GOTO 6300
6140 IF M$="X"THEN LET M% =2:GOTO 6300
6150 IF M$="("THEN LET M% =3:GOTO 6500
6160 PRINT"ADDRESS MODE ERROR IN LINE ";X%:END


6200 LET T%=2:LET M$=LEFT$(S$,1):GOSUB 16000
6220 IF M$="@"THEN LET M% =0:GOTO 6800
6230 IF M$=" "THEN LET M% =2:GOTO 6520
6240 IF M$="("THEN LET M% =3:GOTO 6500
6250 GOTO 6160


6300 GOSUB 10000
6310 GOTO 4500


6500 LET M$=LEFT$(S$,1)
6510 IF M$=")" THEN 6590

6520 GOSUB 12000
6530 IF F%=0 THEN 6630
6540 IF T%=2 THEN 6680
6550 IF H%>0 THEN PRINT"READ/MODIFY/WRITE TO PAGE ZERO ONLY - ERROR IN LINE ";X%
:END
6560 GOSUB 10000
6570 GOSUB 11020
6580 GOTO 4500


6590 IF T%=2 THEN LET M%=5:GOTO 6610
6600 LET M%=4
6610 GOSUB 10000
6620 GOTO 4500


6630 IF Z%=2 THEN PRINT"UNDEFINED LABEL IN LINE ";X%:END
6640 IF T%=1 THEN 6560
6650 GOSUB 10000
6660 GOSUB 11000
6670 GOTO 4500


6680 IF A(Y%)>Q% THEN 6650
6690 IF H%>0 THEN 6650
6700 IF M%=2 THEN LET M%=1:GOTO 6720
6710 LET M%=4
6720 GOTO 6560


6800 GOSUB 16000
6810 GOSUB 6830
6820 GOTO 6560


6830 LET M$=LEFT$(S$,1)                         'EVALUATE A HEX NUMBER IN DECIMAL
6840 IF ASC(M$)<71 AND ASC(M$)>64 THEN LET L%=(ASC(M$)-55)*16:GOTO 6860
6850 LET L%=(ASC(M$)-48)*16
6860 LET M$=MID$(S$,2,1)
6870 IF ASC(M$)<71 AND ASC(M$)>64 THEN LET L%=L%+ASC(M$)-55:GOTO 6890
6880 LET L%=L%+ASC(M$)-48
6890 RETURN



7000 GOSUB 10010
7010 GOSUB 16000
7020 GOSUB 12000
7030 IF F%=0 THEN 7110
7040 IF A(Y%)>Q%+1 THEN 7080
7050 IF (Q%+1)-A(Y%)>128 THEN PRINT"BRANCH OUT OF RANGE IN LINE ";X%:END
7060 LET L%=256-((Q%+1)-A(Y%))
7070 GOTO 7130
7080 IF A(Y%)-(Q%+1)>127 THEN PRINT"BRANCH OUT OF RANGE IN LINE ";X%:END
7090 LET L%=A(Y%)-(Q%+1)
7100 GOTO 7130
7110 IF Z%=2 THEN PRINT"UNDEFINED LABEL IN LINE ";X%:END
7120 LET L%=0
7130 IF O%<16 THEN RETURN
7135 GOSUB 11020
7140 GOTO 4500



8000 LET M$=LEFT$(S$,1):GOSUB 16000
8020 IF VAL(M$)<0 OR VAL(M$)>7 THEN PRINT"BIT NUMBER ERROR IN LINE ";X%:END
8030 LET M%=VAL(M$)
8040 GOSUB 16000:GOSUB 12000
8050 IF F%=0 THEN PRINT"UNDEFINED LABEL IN LINE ";X%:END
8060 IF A(Y%)>127 THEN PRINT"TARGET BYTE NOT IN RAM - ERROR IN LINE ";X%:END
8070 LET O%=B%+(2*M%)
8080 GOSUB 10010
8085 IF B%<16 THEN RETURN
8090 GOSUB 11020
8110 GOTO 4500
```

```
9000 GOSUB 8000
9005 LET J%=L%
9010 GOSUB 7020
9015 LET H%=J%
9020 GOSUB 11000
9030 GOTO 4500


9500 GOSUB 10010
9510 GOTO 4500


10000 LET O%=B%+(16*M%)
10010 POKE(P%+Q%),O%:LET Q%=Q%+1
10020 GOSUB 17042
10030 RETURN


11000 POKE(P%+Q%),H%:LET Q%=Q%+1
11010 GOSUB 17050
11020 POKE(P%+Q%),L%:LET Q%=Q%+1
11030 GOSUB 17100
11040 RETURN


12000 LET L$=""
12007 IF LEN(S$)=0 THEN 12060
12010 LET L$=L$+LEFT$(S$,1):GOSUB 16000
12040 IF RIGHT$(L$,1)=")" OR RIGHT$(L$,1)=" "THEN LET L$=LEFT$(L$,(LEN(L$)-1)):G
OTO 12060
12050 GOTO 12007
12060 LET Y%=0
12070 LET Y%=Y%+1
12080 IF Y%>K% THEN 12110
12090 IF L$=L$(Y%) THEN 12120
12100 GOTO 12070
12110 LET F%=0:LET H%=255:LET L%=255:RETURN
12120 LET F%=1:LET H%=A(Y%)\256:LET L%=A(Y%)-256*(A(Y%)\256):RETURN



13000 GOSUB 16000
13010 GOSUB 6830
13020 GOSUB 11020:GOTO 4500

14000 GOSUB 16000
14010 GOSUB 6830
14020 LET H%=L%
14025 LET S$=RIGHT$(S$,(LEN(S$)-2))
14030 GOSUB 6830
14040 LET Q%=L%+(256*H%)
14050 GOTO 4500

15000 GOSUB 16000
15010 GOSUB 6830
15020 LET A(K%)=256*L%
15025 LET S$=RIGHT$(S$,(LEN(S$)-2))
15030 GOSUB 6830
15040 LET A(K%)=A(K%)+L%
15050 GOTO 4500

16000 LET S$=RIGHT$(S$,(LEN(S$)-1))
16010 RETURN

17000 IF Z%<3 THEN RETURN
17010 LET R%=Q%:LET E%=3
17020 GOSUB 18000
17030 LPRINT TAB(6) R$;
17040 RETURN

17042 IF Z%<3 THEN RETURN
17043 LET R%=O%:LET E%=2
17044 GOSUB 18000
17045 LPRINT TAB(11) R$;
17046 RETURN

17050 IF Z%<3 THEN RETURN
17060 LET R%=H%:LET E%=2
17070 GOSUB 18000
17080 LPRINT TAB(14) R$;
17090 RETURN

17100 IF Z%<3 THEN RETURN
17110 LET R%=L%:LET E%=2
17120 GOSUB 18000
17130 LPRINT TAB(17) R$;
17140 RETURN

17150 IF Z%<3 THEN RETURN
17160 IF LEFT$(P$,1)="*"THEN LPRINT TAB(20)P$:GOTO 17180
17170 LPRINT TAB(27)P$
17180 LPRINT:LPRINT
17190 RETURN



18000 LET R$=""
18010 FOR Y%=E% TO 1 STEP -1
18020 LET I%=R%\(16^(Y%-1)):LET R%=R%-(I%*(16^(Y%-1)))
18030 IF I%>9 THEN LET I$=CHR$(I%+55):GOTO 18050
18040 LET I$=CHR$(I%+48)
```

4. Keeping track of cross references: It is desirable to split the program into the main core and the subroutines. The subroutines do not require specific return addresses to be known for their assembly (RTS and RTI instructions find their arguments on the processor stack). However the main core of the program does require specific addresses for the assembly of subroutine calls, thus the subroutines must be assembled before, or at least simultaneously with, the main core. So the first block of source code to be assembled should contain the most deeply nested subroutines, whose start addresses will then be fixed, and available from the print-out, to be fed back into the second block. The second block will contain the less deeply nested subroutines and the main core of the program. The subroutine start addresses of the first block can then be inserted into the second block by means of EQU operations (see **Fig 12**).

Figure 12

```
.                                       10 DATA *DELAY EQU 187B
.
357 DATA JSR DELAY  Where DELAY is      .
.                   assembled at        .
.                   187BH - becomes:    357 DATA JSR DELAY
.                                       .
.                                       .
```

By breaking the code down into a larger number of sections even a 3.8Kbyte application program can be assembled on a 16K machine without too much trouble. One long summer Sunday I assembled such a program from no fewer than nine previously entered and saved run-time modules – the result at suppertime worked, in that it was correctly assembled, though it transpired that further work was necessary on the program before the project was quite satisfactory.

Conversion for Other BASICs and Systems

1. The use of an integer variable to hold the start address of the assembled code buffer effectively limits the system to using 32Kbytes of RAM. The maximum value of P% is 32767. A further small limitation of usable RAM space is caused by the way that the POKE statements are expressed in lines 10010, 11000 and 11020. These forms make it necessary for the whole of the assembled code buffer to be below 32768, ie. up to and including address 32767, since (P% + Q%) now has a maximum value of 32767.

2. An allied problem occurs in line 12120 where I have used the " " division operation to produce a truncated integer answer, since MBASIC otherwise performs a floating point division and then rounds the answer up *or* down to the nearest integer value.

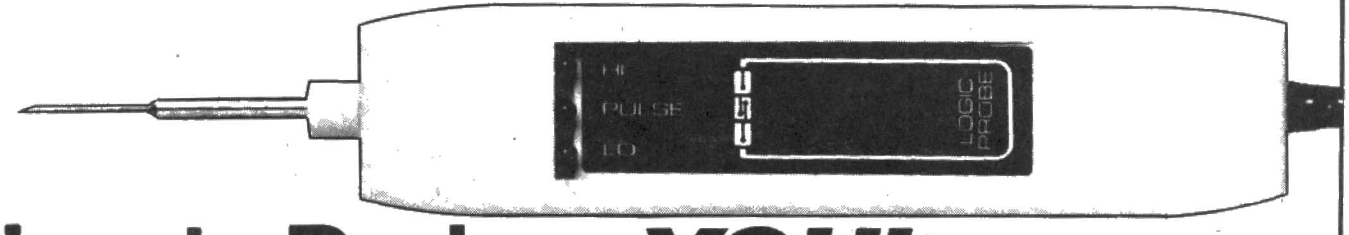
On many systems, it may be possible to rewrite these sections of the BASIC text so as to allow the use of RAM in the upper half of memory space, and to avoid occasional address errors caused by the rounding *up* of the value of H% and the resulting evaluation of L%. I can only leave it to the reader to experiment with their own machine to derive forms of these statements in their BASIC which have the desired effects. There should be no problems encountered in other areas of the routines if floating point variables are used throughout, provided that the requirements of the BASIC POKE statement are met, and a division operation can be provided which always returns an answer rounded down to the nearest integer below the real number answer actually calculated. Something like:

```
........LET H% = INT(A(Y%)/256):
LET L% = A(Y%) - 256*H%.........
```

will probably be available in most BASICs, as a "cleaner" version of line 12120, without resorting to floating point variables.

E&CM

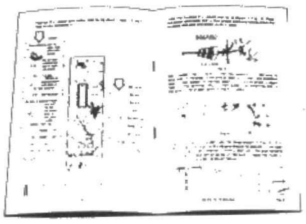
Guess who builds this great



Logic Probe...YOU! for only £14.50

With this easy-to-build Logic Probe Kit from GSC and just a few hours of easy assembly — thanks to our very descriptive step-by-step manual — you have a full performance logic probe.

With it, the logic level in a digital circuit is indicated by light from the Hi or Lo LED; pulses as narrow as 300 nanoseconds are stretched into blinks of the Pulse LED, triggered from either leading edge. You'll be able to probe deeper into logic with the LPK-1, one of the better tools from GSC.



Complete, easy-to-follow instructions help make this a one-night project.

GLOBAL SPECIALTIES CORPORATION

GSC

G.S.C. (U.K.) Limited, Dept. 34Z
Unit 1, Shire Hill Industrial Estate,
Saffron Walden, Essex. CB11 3AQ.
Telephone: Saffron Walden **(0799) 21682**
Telex: 817477.

GLOBAL SPECIALITIES CORPORATION. DEPT 14Z
Unit 1, Shire Hill Industrial Estate, Saffron Walden, Essex.

Name ________________

Address ________________

Inc P&P and 15% VAT
LPK-1 £17.83

I enclose cheque/
PO for £ ________

FREE Catalogue
tick box ☐

Phone your order with Access, Barclaycard or American Express Card No. ________ Expiry date ______

Video Camera Interface

Educational Electronics' camera interface adopts a sensible division between the hardware and software elements of the design. The result is a low cost and highly versatile system. Nick Clare has the details.

The manipulation of digitised images by computer is an area in which considerable interest has been shown in recent years. Applications range from the artistic – such as the high resolution and sophisticated systems used by broadcast TV stations, the use of which can be witnessed each morning on *Breakfast Time* – to the scientific: applications here include robot vision systems and heat loss analysis (in this case the picture is derived from an infra red camera).

The majority of video digitising systems are expensive propositions, as processing 5MHz video signals in real time demands very high speed computation and a large amount of memory. Designs for low cost systems have appeared in the past (the June issue of *Electronics and Computing* carried a low cost system for the Spectrum) but in general the resolution offered has been too low to be of use in many applications. The new video interface from Educational Electronics comfortably breaks the £200 barrier yet offers a very useful performance.

Features

Resolution

220 horizontal x 312 vertical pixels: 64 levels of grey.

Video Input

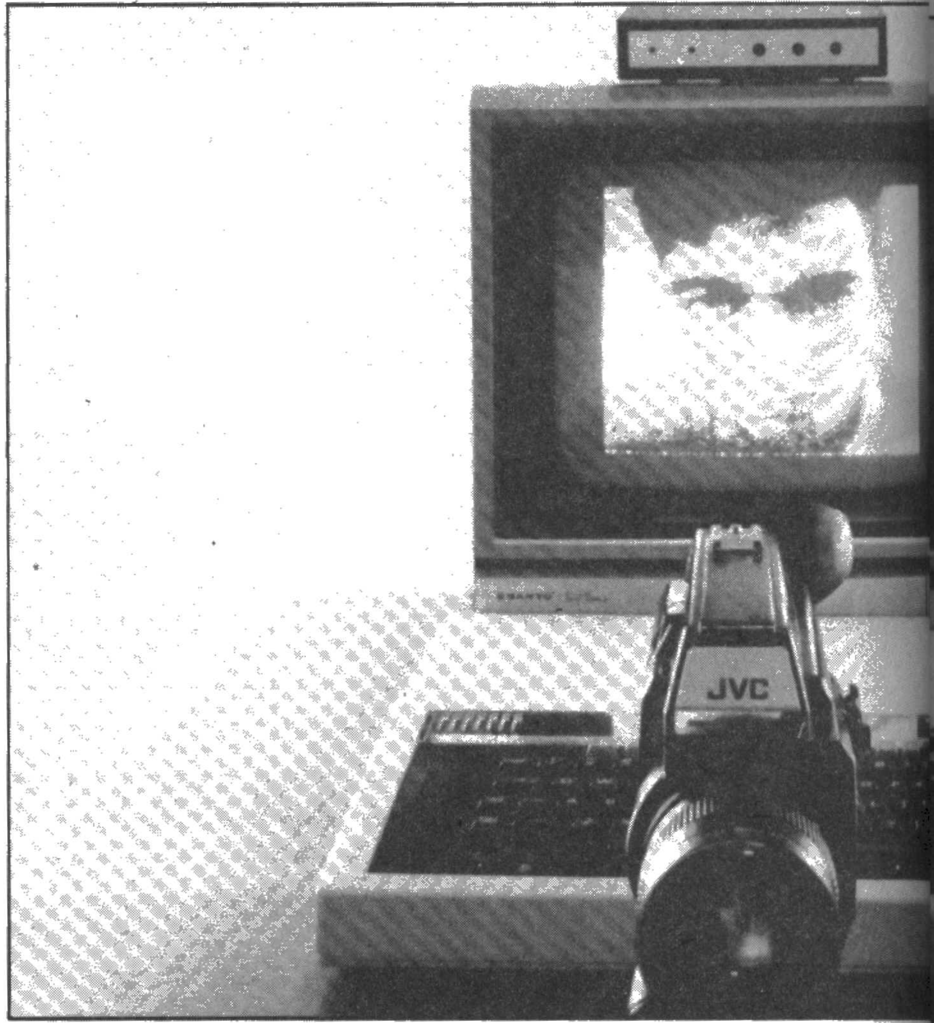
Standard composite – 1 Volt, 75R.

Interface

TTL level, 6 data and 4 control lines via 20 pin IDC.

Power

9-12 volts, 60mA unsmoothed DC.



Performance Parameters

The interface provides a standard 1V, 75R composite video input that accepts signals from a variety of sources such as a video camera, recorder or disc player. The unit produces a digitised image with a resolution of 220 horizontal by 312 vertical pixels with 64 levels of grey, although the ultimate form of the computer generated display will depend upon the hardware of the machine with which the interface is used. At present software has been developed for the RML 380Z or 480Z and for the BBC model B (cassette or disc). The hardware interface only requires 8 TTL input lines to the micro (6 data bits plus 2 signal lines) and 2 control lines from the computer and thus, with suitable software, the interface could be used with a wide variety of systems (software development for the Apple has already begun).

"comfortably breaks the £200 barrier yet offers a very useful performance"

Hardware Highlights

As the block diagram of **Fig 1** shows, the hardware may conveniently be divided into three sections, these being a sync seperator, an A/D converter and a programmable line sync delay. The reason for the line sync delay section is that the interface does not aim to digitise a frame in one pass, the cost of an A/D to accomplish this would be prohibitively expensive, but uses a 'flash conversion technique' to ease the demands put upon the converter and it is here the programmable delay comes in.

Each line sync signal is subjected to a programmable delay and, this delayed signal is used as an instruction that the A/D should begin conversion. The period of the delay thus determines which part of each line is to be digitised – a large delay results in a picture element on the right of the screen being digitised, while reducing the delay will move the digitised element toward the left of the screen. The delay is held constant for one complete field scan and this results in a vertical picture element being converted every 64μS.

Software detects the start of a frame scan, reduces the delay, and thus a vertical element nearer the left of the screen is converted. Digitising a complete frame in this way takes about 4 seconds and during this time it is important that the image is still, any movement will distort the digitised image.

It should also be noted that, while the unit will accept colour video signals, the

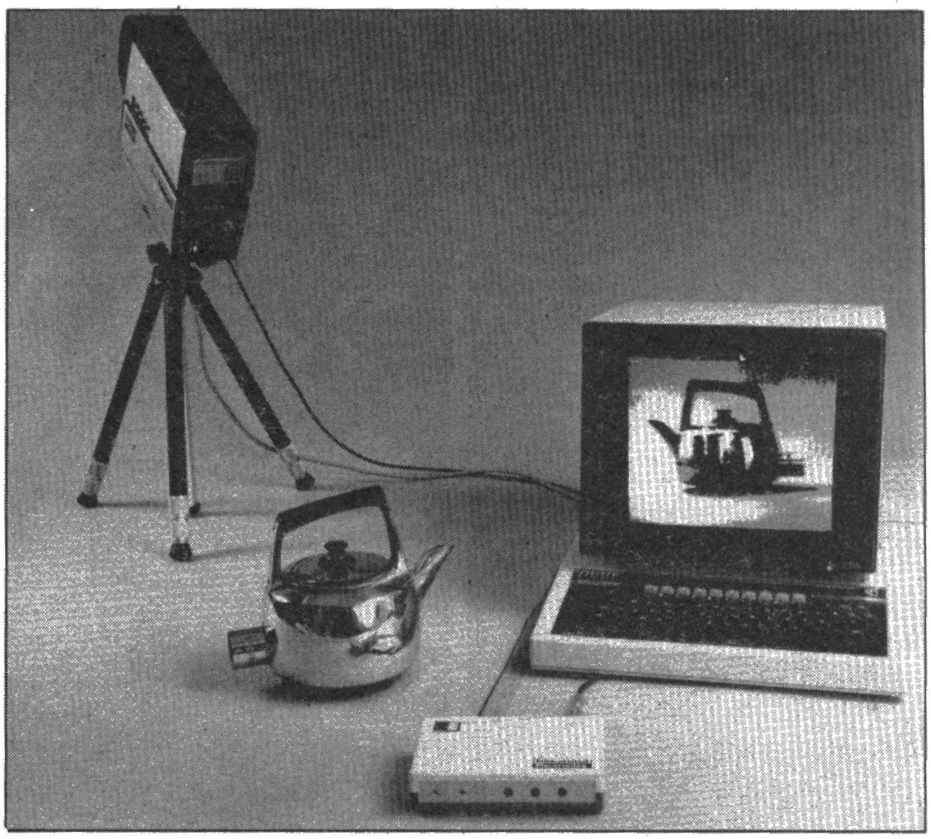
4.43MHz colour subcarrier is ignored, only the intensity signal being processed.

Software Synopsis

From the description of the hardware it is apparent that a major function of the software is to control the line sync delay. In addition, the A/D's output must be captured and stored. In many cases, there will be insufficient memory available to store the information and the computer will not have sufficient speed to process data from one line scan before the next is received.

In a typical application, several of the columns are skipped at the right of the image, by reducing the period of the programmable delay yet ignoring any data. In addition the number of bits read into memory from the A/D converter may be reduced by, for example, reading only alternate lines and averaging the values – this also has the effect of reducing noise.

As the software is in full control over the column position to be digitised it is possible to look at only a few vertical picture slices. This technique is particularly useful in applications such as movement detection where comparison of a few columns near an object's edge will give information about direction and speed of movement.


The camera interface in use with a BBC micro and a standard black and white TV camera.

In Use

The unit was demonstrated with both a BBC computer and an RML machine. The RML, with its more sophisticated video circuitry, gave more versatile results and an example of the display is shown in **Photo 1.** Results with the BBC were also impressive, as shown on the front cover of this issue.

The digitised image can be stored on disc for later retrieval or processing. The colour values of each level of grey can be selected from a palette of colour that varies with the specific computer being used for the display.

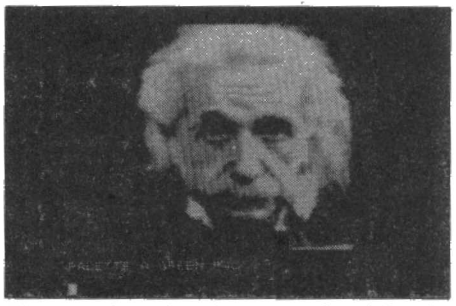


Photo 1. A typical image produced by the system.

By careful selection of the colours, object boundaries may be highlighted and this is a valuable facility in areas such as cell counting. Object recognition and photo elastic stress analysis etc. (Educational Electronics offer a design service that can provide specific software packages).

The software is written in such a way that the various section of it are accessible to the user and could be readily tailored for various applications.

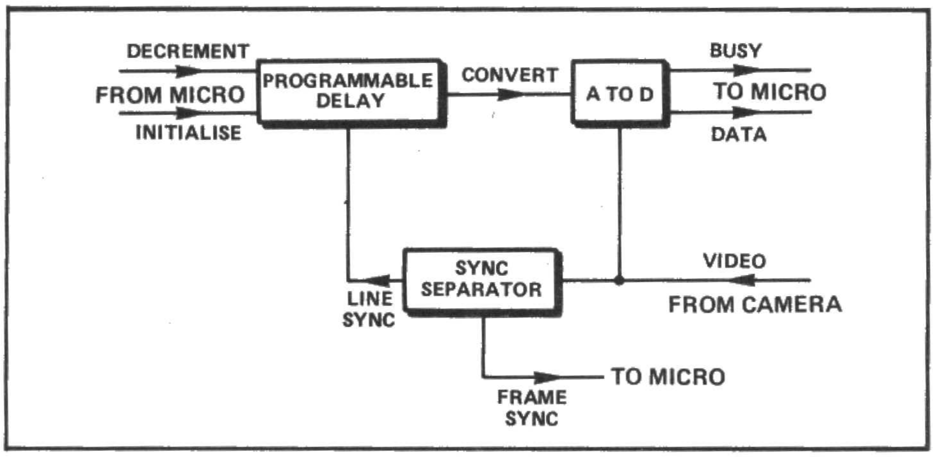


Figure 1. Block diagram of the video camera interface.

Conclusion

The interface, at £174 excluding VAT, provides a low cost way in which the field of pattern recognition can be explored. The hardware is well built and the software is both versatile and bug free.

Educational Electronics, 30 Lake Street, Leighton Buzzard, Beds LU7 8RX.

E&CM

PORTABLE PRINTING POWER
The Brother EP22 reviewed

An electronic typewriter with an RS232C interface, 2K of memory, a fine key action, adjustable LCD, battery operation, and weighing in at 5.3 lbs in its case: this is the stuff that dreams are made of. But the Brother EP22 is a reality with the down to earth price of £170. Sammy Roth checks out the goods.

Before anyone dashes out to make their purchase, it should be said that the EP22 has its faults. The print quality is not adequate for every purpose, the correction facility is limited, and the use of either thermal paper or ribbon cartridges involves substantial secondary costs. These points will be dealt with below; first let us examine the multitude of interesting and useful features crammed into the one powerful package.

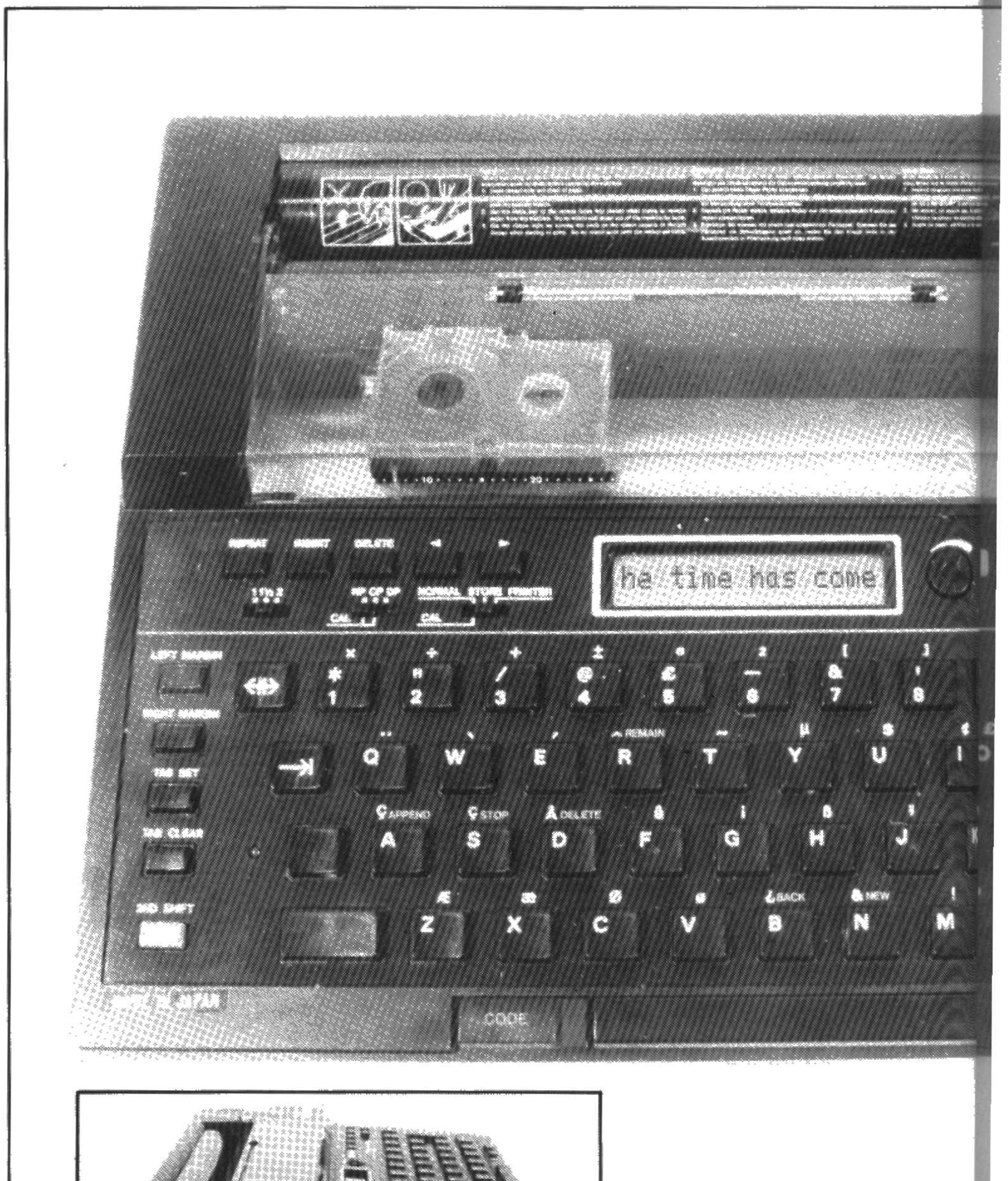
The Keyboard
The EP22 has a total set of 132 characters on 44 keys. Using shift and second shift a comprehensive range of mathematical, financial and continental symbols can be obtained as well as the usual alpha/numerics. These keys also fulfil a number of memory functions when pressed in conjuction with the code key. In addition there are a further 16 typing function keys controlling margins, tabulation, shift, carriage return, up/down, backspace etc; switches to control line spacing, print mode and correction mode; repeat, insertion, delete and cursor keys, and of course calculator keys – the EP22, of course, has an inbuilt calculator!

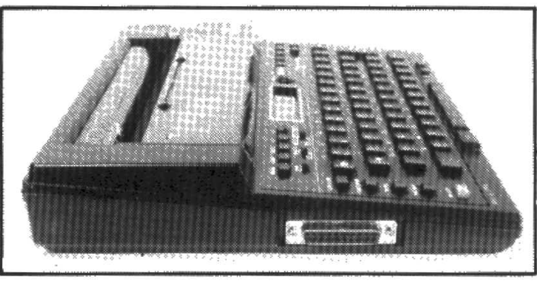
Despite the multiplicity of character and function keys, this is a very easy machine to use, in fact a real joy. The keys have a light, sensitive touch and are spaced an adequate distance apart. It is easy, if not preferable to type with the machine on your lap. The space bar is wide and long, and the shift keys are correctly spaced – unlike many electronic typewriters and computers in which they are often placed too far to left or right.

Typing Modes and Memory
Three typing modes are available. The first, non-print, is used when typing data into memory, and the print mechanism is de-activated. Direct-print mode prints out the characters as they are typed, and correction-print gives a 16-character delay; that is, the user has the opportunity to make deletions and insertions to the text visible on the 16-character LCD before it is printed out – a vital facility for the inaccurate typist.

In non-print/store mode, characters are stored in the 2K (in fact just under 2K) memory, which translates into about 1 A4 page of text. It is possible to append text and to delete the whole of the last line entered, and of course ammend the previous 16 characters entered (those on the screen). But remember this machine is **not** a word processor. it is impossible to ammend any other than the last line entered, so if the user intends to print out 20 copies then he/she must keep a wary eye on the display. There is one, clumsy, method of correcting text once entered into the memory; this is to stop the printing process (can be done at the end of each full line) and manually type in the flawed line with its correction. This facility also enables the user to type in variables such as addresses etc.

The Printing Head
For £170 you cannot expect a daisy wheel, but even so the dot matrix pin device used in the EP22 is a disappointment. With the





Above, the Brother EP22 'Electronic W A is electronic. Left, the RS232 interfac machine.

carbon ribbon cartridge it is necessary to use very thin, highly glossed paper (Brother recommend 50gm.). Such specifications result in a cheap looking production, and I found photocopy paper – which is thicker and less glossy than the stock provided with the machine – to give a better presentation. The text thus printed is of irregular strength and

"no problems as a printer for the BBC"

difficult to read. Add to this the speedy depletion of the ribbon (one cartridge prints perhaps 10-20,000 characters) and you soon decide to resort to the more expensive option of thermal paper. By removing the cartridge the Brother becomes a dot-matrix thermal printer giving a far higher quality, legible type over the full 75-column page width (users of 80-column text screen computers should note this figure!). One further disadvantage – an irritating and perhaps unnecessary omission – is the lack of descenders on the characters.

The EP22 as a Printer

Now we come to the meat. The EP22 is equipped with a 25-pin RS232C interface, neatly hidden from innocent eyes by a detachable cover. 75 and 300 baud rates are available. Setting up the machine to receive data is simply a matter of switching on, selecting the baud rate by pressing the return key, and pressing the CONT key. The data which is stored in the printing buffer (78 bytes) is cleared when this key is depressed. Even if the STOP key is depressed to assume the off line state during printing, the data remains in the printing buffer. Therefore the remaining data is printed out when the CONT key is depressed, even if the host computer is turned off.

For those interested, the manual gives comprehensive information about ASCII codes, control code functions, printer specs., signal levels, pinouts, and the printing buffer. A separate manual gives all necessary connection and addressing information for the following computers: Atari 400/800; Commodore 64 and VIC 20; TI99/4A; Epson HX-20; NEC PCs 8201 and 8801; Sharp PC-1500; Sord M223 (III); Tandy TRS-80; and the Colour Genie.

Readers will immediately observe two notable absentees from this list: they are the BBC Micro and Sinclair Spectrum. Needless to say these were the models we chose to test out the Brother's performance!

First the BBC. There were no problems whatsoever at either 75 or 300 baud rates. The printer secreted all the information entered into the BBC. The correct connections are printed out above.

With the (300 baud only) Spectrum it was a different story. Despite checking and re-checking the connections, and a speedy despatch to a component company for the correct connecting cable, our efforts were fruitless: the printer remained utterly silent. Readers should not, however, imagine that this is necessarily the fault of the EP-22. A probe applied to the RS232 output of Interface 1 produced no trace on the oscilloscope, and it is therefore possible that the fault lay within the Interface. That said, the verdict will have to remain open, and until the case is proved one way of the other the value of the connection diagram printed below will be academic – so watch this space.

Miscellaneous Data

Power supply is by either four 1.5V batteries, or via a 6V AC mains adaptor (an optional extra). The weight given for the Brother (5.3 lbs) includes the batteries: a significant proportion of the sum. Therefore the EP-22 is truly portable. It is roughly the size of an A4 ring binder file. It has its own case, which fits onto the top of the machine, and is easy to carry.

Printing (at a speed of 17 characters per second) is virtually silent, except for an audible clank as the carriage returns. Typing can be done in total silence in non print/memory storage mode, which makes the Brother absolutely ideal for those wishing to work on planes, trains, or any other place where it might conceivably by necessary to type information in an organised, legible form.

The Brother is a remarkable machine, and an excellent buy for home computer users who want a cheap A4 printer with a free typewriter to boot. Nevertheless the fact must be borne in mind that headed notepaper (unless photocopied) or any kind of thick cartridge paper will produce useless results, and, that letter quality hard copy cannot be produced under any circumstances. Despite these drawbacks, Brother has shown up every other cheap printer manufacturer as an incompetent, and hopefully the EP23(?) will have an improved print head.

The Brother EP22 is available from Lowe Computers, Bentley Bridge, Chesterfield Road, Matlock, Derbys DE4 5LE.

E&CM

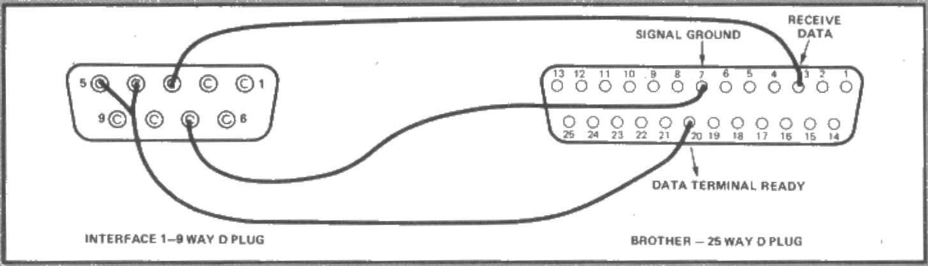


Figure 1. Connections between EP22 and Spectrum interfaces.

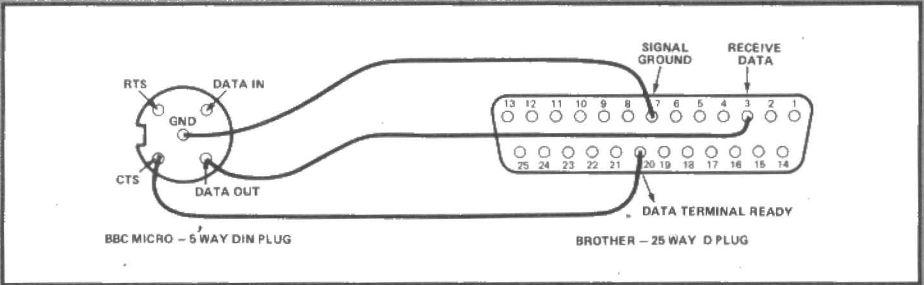
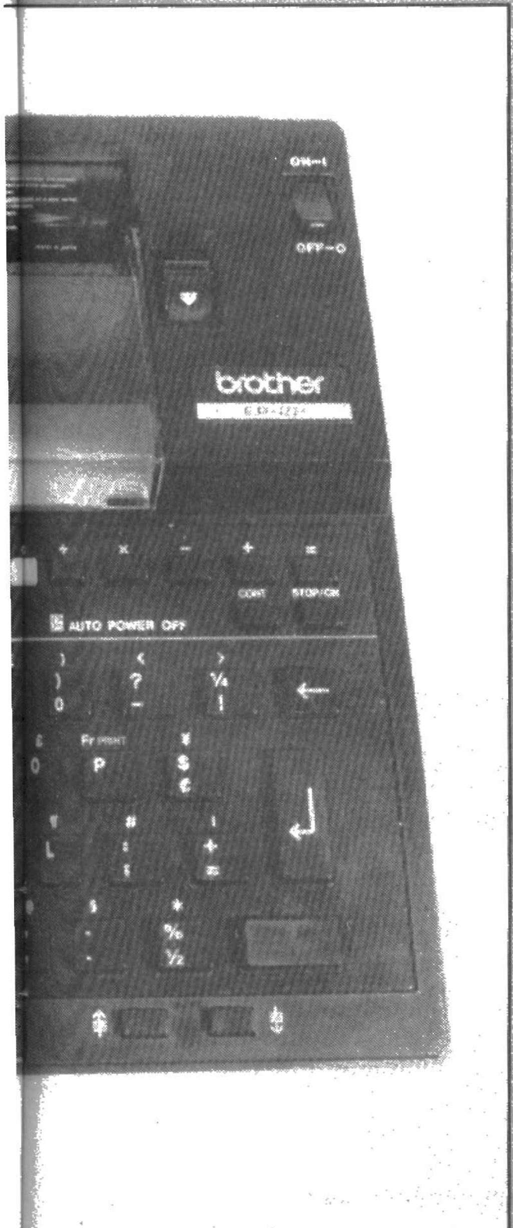


Figure 2. Connections between EP22 and BBC micro interfaces.



. All carriage rotation and return movement clearly visible on the left hand side of the

INTO 1984

Mike James takes a retrospective look at the 1983 computer market and speculates on what 1984 may bring.

The year of 1984 is, in fiction at least, not a good one for computers and electronics – if you happen to be an unfortunate citizen that is! The real 1984, however, promises to be a good year for computers but the difference is that with luck it should be a good year for computer users as well. Personal computing is still changing at a pace that makes it difficult to keep up with what is going on in the present, let alone in the future. However, it is fun to try and guess what is going to happen and it can even be useful to speculate on where the constantly developing technology is taking us.

Past Hardware

The year 1983 was surprisingly quiet from the point of view of new machines, prices fell but there were no great hardware innovations. About the only real hardware breakthrough in '83 was the advent of the portable battery driven family with the Epson HX20. The growing numbers of such machines is due to the development of high density CMOS chips as opposed to the current-guzzling N-MOS chips normally used for microprocessors. This trend is continuing with various chip manufacturers announcing CMOS versions of the familiar 6502 and Z80 and of the newer 16-bit devices such as the 68000. When these chips are available in more than sample quantities (during '84) there is a good chance that all personal computers will be battery-powered, smaller and cooler!

Apart from the portable boom the micro world seemed to be concentrating on producing low cost Spectrum-like machines. The early part of the year saw the ORIC actually reaching the hands of customers and more recently the Aquarius has made its appearance (possibly a brief one). The Sord M5 was just one of the Japanese machines that tiptoed onto the scene without shaking the foundations of the UK market. It is something of a continuing mystery why the now long-predicted Japanese take-over of the personal computer has never materialised.

"The only hardware breakthrough in '83 was the battery driven computer".

Perhaps this has something to do with the way that UK-designed machines have been able to take advantage of the custom chip design and production facilities of Ferranti. Whatever the reason the lower cost end of the market is still dominated by UK manufacturers – perhaps this will change in '84?

The market for computers certainly became a much more unfriendly place from the point of view of manufacturers. In some cases this was due to rather too much success and rather too little initial funding, in other cases it was clear that the product was simply not up to scratch. So far the financial injuries have been mostly of a temporary nature. The Newbrain was one of the first machines to run into problems but now it looks as though it has been taken up by another firm and will be relaunched as a business machine (how anyone could have thought that it could compete with machines such as the Spectrum is difficult to see). Dragon Data has had a successful but difficult time trying to find enough cash to sustain its development. A measure of just how tough things are can be gained from the recent news that the firm behind Jupiter Ace (the Forth machine) has gone broke. All this suggests that 1983-1984 is not a good time to issue a new computer unless you have something really special at a really special price! However this hasn't stopped firms from trying. The ZX add-on manufacturers Memotech have gone the whole way and produced a machine of their own and 1984 should see how it fares. The biggest machine news however must be the small addition to the Acorn/BBC family of micros – the Electron. This is one machine that enters the market with a head start!

In the States things have been just as tough if not more so. With the introduction of the IBM PC and Apple's latest LISA the top end of the market seems to be hotting up. At the lower cost end, firms such as Atari and Texas have reported losses on their personal computer operations. Both firms introduced new models and cut prices during the year but at the moment it looks as though this was too little too late. Apple also introduced a replacement for the antique APPLE II – the APPLE IIe. Currently the only product that Apple has that looks at all interesting is LISA and this may be a little too novel to succeed (see the discussion of software later).

What these troubled times have meant for the personal computer buyer is simple – lower prices. 1983 saw, not only the under 100 pound colour computer, but the much less than 50 pound black and white machine! The price cutting is not all good however and it might leave some owners with machines that no-one wants to support.

In the area of peripherals the biggest single event is the final appearance of the Sinclair Microdrive. When the Spectrum first appeared there were few people who would have thought that it could take quite so long for the micro drives to be produced. Now they are with us the only problem is that there aren't enough of them. Along with the micro

drive come two new interfaces. Interface 1 gives the Spectrum a serial port and a limited networking capability. Interface II gives it the ability to use ROMs and switched, ie non-analogue joysticks. These peripherals are likely to be important but only when there are plenty of them available – which is not now going to be during 1983.

What '84 Might Bring

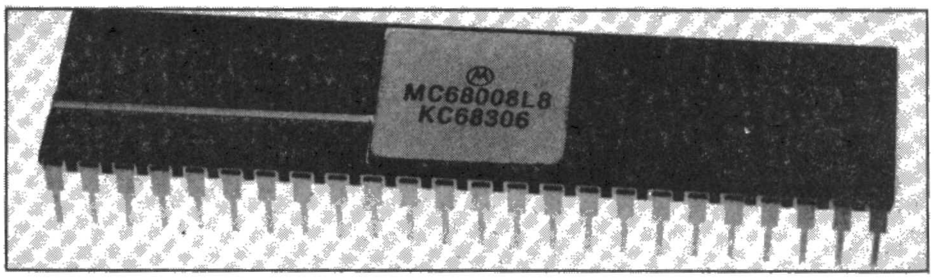
The steady changeover from eight- to 16-bit micros cannot fail to gain even more momentum next year. At the moment the price of 16-bit microprocessors is too high to make them candidates for low cost personal computers but IBM has its 8086 based small machine. The impact of this machine, nicknamed the 'Peanut' will be all the larger because it is likely to be IBM PC compatible. IBM are hardly taking over the personal computer world overnight but they are steadily gaining ground from the traditional market leaders in the USA. Apple will probably produce a new low cost machine but it will have to be cheaper and offer more than the APPLE IIe to offer any hope of success.

Apart from these two obvious moves from the USA's big two it is difficult to predict new machines in the coming year. The main reason for this is that there are very few gaps in the market. It is stating the obvious to say that the time is right for Sinclair to announce a more serious machine to be compatible with and marketed alongside the Spectrum. Dragon Data do have fairly firm plans to issue one, may be even two business machines in '84. Rumour also has it that Acorn are planning a full-function business machine to complete their coverage of the market. Apart from this any other manufacturer will have a fairly hard time convincing people that their product is better than existing models.

"The changeover from 8 to 16 bit micros will gain momentum in '84".

The use of a 16-bit micro in a reasonably priced personal machine is something that I will be hoping for in 1984 but apart from IBM's offering the prospects look poor. One route to more computing power that should materialise next year is the range of second processor add-ons to the BBC Micro including the 32-bit National chip! When 16/32 bit machines do become common it will mean a considerable change to the sort of applications that are possible with a personal computer.

A new range of peripherals should find their way into new systems next year. In particular the much talked-about 3" disc should finally manage to get its formats and price sorted out. Unlike the Sinclair Microdrive which uses a continuous loop of tape, the 3" disc is a true disc. Its main advantages are that it can hold as much data as a 5" disc, it should be cheaper and more robust. Existing 5" disc users will welcome the announcement of versions of the 3" disc that will be electrically compatible with their standard 5" interface. Another important development in the disc market is the removable Winchester. Currently disc manufacturers are announcing combinations such as 5Mbyte fixed plus 5Mbyte removable disc drives. For the slightly more upmarket machines one of these drives could provide all of the storage that they would ever need and would fit into the space of a dual 5" drive.



Peripherals seem to be the most exciting prospect in 1984 in other ways that are more relevant to readers of *E&CM.* Next year should prove to be the first time that the prices of what might be loosely termed 'robots' reach reasonable levels. Already motorised 'buggies' are on the market for the BBC Micro and the Spectrum and low cost robot arms should follow in 1984. Small video input devices are becoming available at affordable prices. What this sort of hardware will be used for is something worth speculating on! It cannot fail to be fun but it might take some time before any really useful applications are perfected! This, however, is a challenge worth taking up in the next twelve months.

Software

It is very difficult to decide what the major software developments have been or to predict what the next ones will be. The reason for this is that software develops over a much larger time scale than hardware. This should be obvious from the fact that old languages such as Fortran are still being used! Software tends to develop slowly and without the neat, clean breaks that typify hardware development.

So far it is possible to list three generations of software. The first generation was the highly technical systems and applications software that could only be used by computer experts. This software is best characterised by saying that it tended to do things in the way that the computer wanted them done. It seems safe to say that we are well past first generation software and well into the second generation. Second generation software is meant for use by normal mortals. In the jargon it is "user-friendly". This doesn't mean that users don't need training to use second generation applications packages – they most definitely do – but the software makes great efforts to do things the way the user would like it to be done. Most of the software that we use at the moment falls into the second generation category. Third generation software is just beginning to appear. It can be distinguished from the second generation user-friendly software by its ability to 'fit in' to the way that the user does things. Third generation packages incorporate enough artificial intelligence methods to allow the user to instruct the program what is needed and let the program sort out how to achieve the result. The best known example of third generation software is the expert system but many natural language understanding programs are likely to appear during the coming year. The impact of good second and third generation software should increase the number of people who can make use of computers and a widening range of applications.

Systems software will also be a focus of attention next year with the two best known microsoftware firms, Microsoft and Digital Research engaged in almost open warfare with their rival operating systems, MSMOS and XENIX and the CP/M family respectively. This battle is in fact a reflection of the wider UNIX versus CP/M war. The trouble is that UNIX and other multi-user operating systems are very expensive at the moment and this tends to limit their interest to the personal computer war. At the moment most people are backing UNIX as the CP/M of the future but Digital Research are working on such a range of new products it would be foolish to write off CP/M at this early stage. Concurrent CP/M 86 is a single-user multi-tasking operating system that is more economical on memory than UNIX. Personal CP/M is planned as a competitor to the other new style operating system that can be found running on Apple's LISA. The idea behind these new operating systems is that, instead of using commands to manipulate data and files, a graphics model of an office is presented and operations are in terms of manipulating familiar objects. For example, erasing a file is represented as throwing it in a waste paper bin! Whether this sort of thing has anything to offer the personal computer user is difficult to say but if it works it will revolutionise the way computers are used in business.

Getting Better?

Will 1984 be better for computing than 1983? From the discussion above the indications are that there will be fewer new machines in '84 but many more interesting uses of computers. With a little luck next year could be the opportunity to settle down and concentrate on using your microcomputer rather than worrying about whether a new machine might do things better!

E&CM

Experimental science with the BBC micro

David Knaggs shows how the BBC microcomputer can be used as a versatile scientific instrument, and explains one application: the measurement of the coefficient of resistance.

The trend towards greater numbers of microcomputers in schools seems set to continue apace in the forseeable future. As interest in the applications of these versatile machines spreads to departments other than mathematics, and with the greater machine availability being encouraged by both the local authority and Government, it is to be hoped that the science departments will make much greater use of the hardware than is often the case at present. The power of the micro-computer, both as a high speed calculator and as a data gatherer can be put to most effective use in experimental science where the speed of the computer can remove the drudgery of routine calculations and repetitive measurement to allow concentration on the essential points.

The purpose of this article is to describe an application of the BBC microcomputer to a standard Physics experiment, namely the measurement of the temperature coefficient of resistance of three different materials. The advantage of the BBC micro in this type of application is that it has, in the Model B version, a four channel Analogue-to-Digital Converter (ADC) fitted as standard.

The selection of suitable transducers, together with the inclusion of the relevant calibration data in the driver software, will allow the BBC machine to give digital representations of quantities such as voltage, current, resistance, temperature, force etc. directly from the experiment. The scheme described here will allow sufficient data to be collected to give values for the temperature coefficient of resistance of samples of Copper, Manganin and for a Thermistor, which may be expected to show a variation from a positive value, through an almost zero value to a negative value.

The data collected from this experiment is used in the standard formula for the calculation of temperature coefficient of resistance ie:

$$\text{Alpha} = \frac{\text{Rt} - \text{RO}}{\text{RO} * \text{t}} \text{ per K}$$

where RO = resistance at zero Centrigrade
Rt = resistance at t degrees Centigrade
t = temperature rise above zero degree Centigrade
Alpha = temperature coefficient of resistance

The standard method used in the determination of Alpha is to immerse the material under test in an oil bath, the temperature of which is then progressively raised. At suitable intervals the resistance is measured, often using a bridge method, until the temperature range under consideration has been covered. This type of method has been adopted for this experiment with the advantage that the data can be collected quickly and completely automatically, by the computer. The experimental arrangement used is shown diagramatically in **Fig 1.** The oil bath was heated using an immersion heater of the type which is commonly found in school science departments and which operates from a 12 volt supply. The oil was stirred continuously by a paddle, driven from a low voltage motor, to establish as even a temperature distribution as possible. The duration of the heating cycle was controlled by the computer, using the arrangement illustrated in **Fig 2.** The line PBO from the user port, was connected to the base of a transistor, which had a relay coil connected as its collector load so that when PBO is made high the transistor is switched on and the relay contacts close to complete the heater circuit. The heating cycle was fixed at 50 seconds for this experiment as this was found to allow a reasonable temperature difference to be produced for each cycle.

The measurement of temperature was achieved with the aid of the circuit shown in **Fig 3** which is based around the 555 timer circuit. For this application the 555 is connected as a free running oscillator whose frequency is determined by the resistance and capacitor values used in the timing network. This frequency is varied by the effect of temperature on a bead thermistor which is used as one of the elements in the timing network. The use of a bead thermistor is essential as a very low heat capacity is required to allow the system to have a rapid response to changes in temperature of the surroundings. As the temperature of the thermistor increases, its resistance will decrease and the oscillator frequency will rise accordingly. The output waveform produced by the oscillator is a square wave which is suitable for input to the computer. In order to obtain a value for temperature, the oscillator frequency must be measured and converted from calibration data which is stored in the program. The calibration data used in this example was obtained by measuring the number of pulses arriving in half a second, for a number of different temperatures. This data was then used graphically (see **Fig 4**) to give conversion factors which were included in the program for use in the calculation of temperature. It was found during this procedure that the

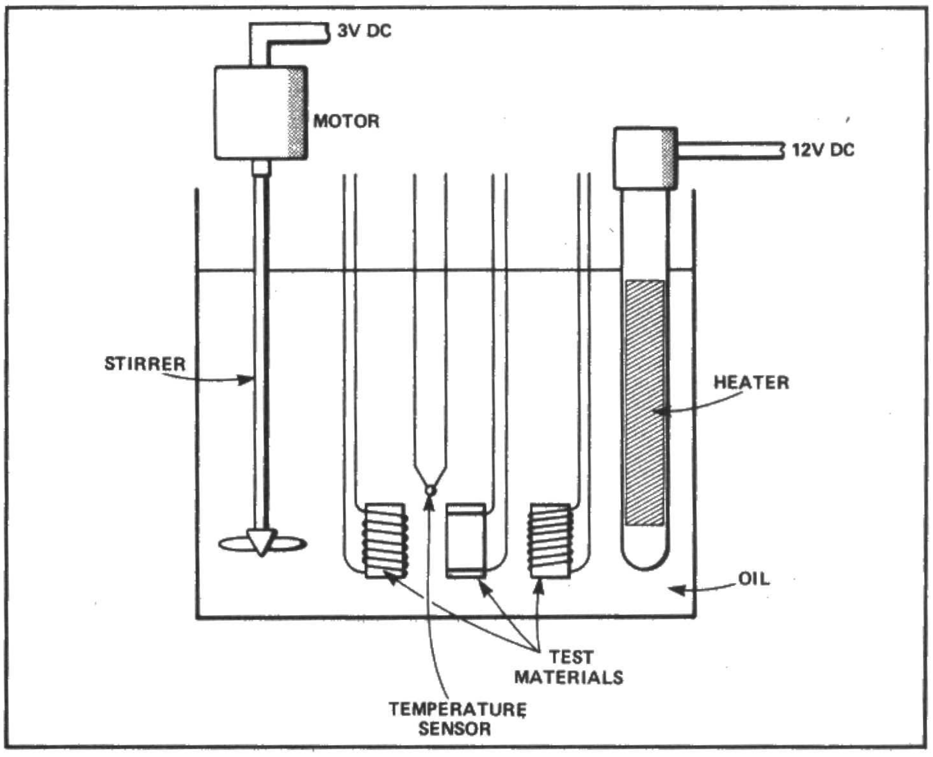


Figure 1. Insertion of test materials into oil baths.

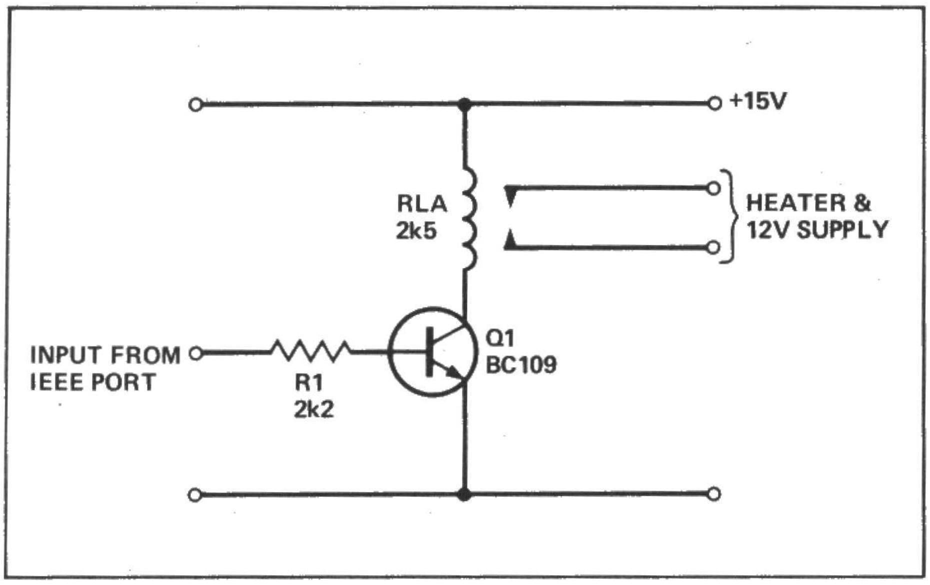


Figure 2. Heating cycle control circuit.

variation of frequency with temperature was not linear but that the calibration could be carried out by using a series of straight line segments to represent the frequency – temperature relationship. There would seem to be little loss of accuracy in making this approximation.

The experiment uses all four channels of the ADC to measure voltages at the required points in the circuit shown in **Fig 5.** The main power supply used operated at 5 volts but was connected to the circuit via a series resistor in order to give an effective supply which was compatible with the reference voltage used by the ADC ie. approximately 1.8 volts. The specimens and their series resistors were selected to be as nearly as

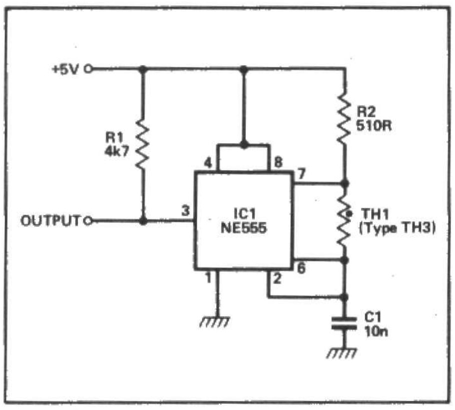


Figure 3. Temperature measurement circuit.

possible of equal value, in order to reduce any errors of measurement by making the two voltages initially equal. As the three branches of the circuit were connected in parallel the supply voltage for each was the same and thus only one determination of this voltage was required. In the prototype, the resistors, oscillator and relay were mounted on printed circuit board, while the test specimens and the temperature sensing thermistor were mounted on a separate board and the two were interconnected by a multi-core flying lead to allow the specimens to be easily placed in the oil bath. The connections to the ADC were taken from the junction of each specimen and its series resistor, as well as from the supply lines. These connections give access to all the voltages in the circuit and also monitor the supply voltage so that any variation can be allowed for in the subsequent calculations.

The operating program was required to provide both the control functions required to generate the correct sequence of events, to measure and store the data in an easily retrievable form and to manipulate the data to yield the final result. The control and calculation functions are worthy of some detailed consideration.

Control

The major requirements of this system are an ability to switch the supply to the immersion heater and to access the ADC at the correct times and in the correct order.

The heater switching is achieved by the use of the POKE statement which can change the state of any of the lines in the user port. The method of using this type of instruction is to first set up the port in output mode by setting the relevant bits in the Data Direction Register (DDR) to logic 1 levels. In this case only one line is required as an output which results in the statement:

?&FE62 = &01

where &FE62 is the address of the DDR. As line PB0 is connected to a switching transistor, making this line high will switch the transistor on and complete the heater circuit. This effect is achieved as follows:

?&FE60 = &01

where &FE60 is the address of the Data Register in the output port. Loading this address with zero will have the effect of switching the transistor, and thus the heater, off.

Measurement

The measurement functions are required in two areas, namely in the determination of temperature and in the measurement of the various voltages which are required for calculation purpose at a later stage.

The measurement of the number of pulses received by the computer in a pre-determined time interval is achieved by the use of timer/counter T2 which is included in the 6522 VIA. Timer T2 is a 16-bit device and when correctly initialised, will count pulses which appear on line PB6 of the user port. The registers are seen by the computer as memory locations and these are used as follows:

a) load the Auxilliary Control Register with 32 (ie. bit 5 is made a logic 1)
ie. ?FE6B = &20 ACR loaded with 32 (dec)

b) load the low order register with 255
ie. ?&FE68 = &FF low order register loaded with 255 (dec)

c) load the high order register with 255
ie. ?&FE69 = &FF high order register loaded with 255 (dec)

Any pulse now detected on PB6 will cause these register contents to be decremented by one and at the end of the counting period the actual count is found by subtracting the final register contents from the original value. This result is then used in conjunction with the calibration data to yield a value for the temperature.

On completion of the counting period the register contents are read with a 'Peek' statement eg:

Z = ?&FE68
Y = ?&FE69

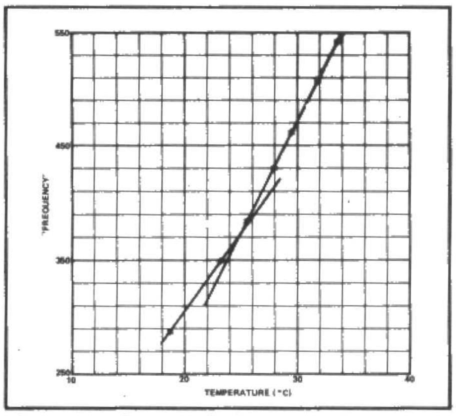


Figure 4. Graph of the conversion factors included in the program.

TEMPERATURE (°C)	AVERAGE 'FREQUENCY'
18.7	288.6
23.1	348.9
25.5	387.0
27.9	429.7
29.5	461.0
31.8	508.0
33.5	541.8

It is standard practice to read the low order register first as this is subject to the more rapid rate of change.

As it is unlikely that the number of pulses counted in a given period will always give the same value, a limit was set so that a maximum of five counts was the allowed variation between any two consecutive measurements. The calibration data used suggested that at a temperature of 25 deg. C

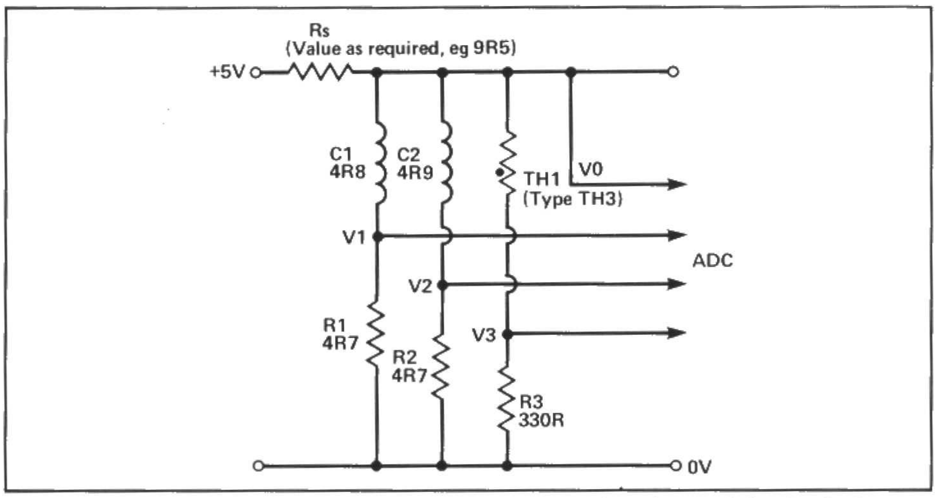


Figure 5. All four channels of the ADC are used to monitor the voltages of the circuit shown.

the maximum error that would be introduced would be of the order of five parts in 380.

The ADC enables voltages to be represented in digital form with great ease and at high speed. The four channels were used here to measure voltages as follows:

Channel 1 – Supply voltage
Channel 2 – Copper specimen
Channel 3 – Manganin specimen
Channel 4 – Thermistor

The results of the conversions of the ADC were used to give absolute voltages as follows:

eg. V(2,4) = (ADVAL(2)/16)*(1.8/4095)

which is the voltage across the copper specimen on the fourth cycle of the experiment. The factor of 16 is included to reduce the 12-bit ADC output to the more usual 8-bit representation, while the second term gives the conversion from ADC output to voltage.

Calculation

The data collected during the experiment requires simple manipulation to produce a value for the temperature coefficient of resistance.

i. The resistance of each component is calculated for each cycle of the experiment as follows. The current flowing in each branch of the test circuit is found first from the measured voltages and the value of the series resistance, using Ohm's Law.

ii. This current is then used in conjuction with the voltage across the specimen to give a value for the specimen resistance.

iii. The temperature for each cycle is found by first comparing the measured 'frequency' with the value at the 'break point' as indicated by the calibration data, and then using the gradient of the appropriate segment of the graph to give a value for the temperature in degrees Centigrade.

Considering the complete set of results for each specimen, a value of the resistance at zero degrees Centigrade was then found. This assumed that over the temperature range of the experiment, the relationship between resistance and temperature was linear and so the resistance at zero was given by extrapolation of a line between the first and last measurement. Using the information outlined here the value of Alpha was calculated. On completion of the experiment and calculations the data collected is displayed in tabular form along with the values obtained for the resistance at zero and the value of Alpha. This is repeated for each material with a delay included to allow inspection of the results or for details to be copied from the screen.

At the end of the presentation of the results, a 'menu' of output options is given which allows the user to select the method most appropriate to his/her situation.

Next Month – the software.

E&CM PCB SERVICE

April 1983	
TV to RGB Conversion	£2.70
July 1983	
Power Control For Micros	
Relay Board	£2.02
DAC Board	£1.77
Stepper Motor Driver	£1.59
BBC Sequencer Interface	£2.10
August 1983	
Oric Output Port	£2.10
Spectrum Sound Board	£2.20
September 1983	
BBC Darkroom Timer	£1.15
Cassette Signal Conditioner	£1.23
ZX81 Sound Board	£3.77
October 1983	
Spectrum Effects Box	£1.71
Cassette Signal Conditioner	£1.23
BBC EPROM Programmer	£5.12
November 1983	
Lie Detector Interface	£1.88
Microcontroller	£2.13
ZX Light Controller	£4.28
December 1983	
BBC Sideways RAM	£4.98
Electron A/D	£2.91
January 1984	
Electron I/O Port	£2.32

HOW TO ORDER

List the boards required and add 45p post and packing charge to the total cost of the boards. Send your order with a cheque or postal order to:

ECM PCB Service, 155 Farringdon Road, London EC1R 3AD.

Please supply the following PCBs: ....

....

Post & Packing 45p

TOTAL £ ______

Signed .... Date ....

Name .... (please print)

Address ....

....

....

PLEASE ALLOW 28 DAYS FOR DELIVERY

Introducing Z80 and 6502 Vectors

PART 2

Following on from last month's foray into the unknown world of BBC and Spectrum vectoring, Adam Denning demonstrates a few practical examples of the technique.

Having shown how to vector the Spectrum's Mode 2 interrupts into useful utility routines, it seems fit to explain how to extend the BASIC ROM to include and act upon other commands of the user's making. Unfortunately, this is only plausible on Spectrums fitted with Interface 1, the Microdrive add-on, as otherwise the on-error vectoring (see below) simply does not occur.

In their laudable efforts to maintain compatibility and low prices, Sinclair have caused the 8K shadow ROM contained within Interface 1 to be called by a very devious technique. As most Spectrum programmers will by now know, the BASIC error handling routine revolves around that piece of Z80 exotica, the restart. RST 8 in fact handles the errors, and on an unexpanded Spectrum this causes the message defined by the byte immediately following the error restart call to be displayed and the BASIC program currently running to terminate.

Error Vectoring

Now, on a Spectrum with Interface 1 fitted, any error causing an RST 8 is re-examined by the Interface ROM after the standard ROM has rejected it for whatever reason. Therefore, the strange and unwieldly Microdrive commands (SAVE *"m";1; , LOAD *"m";1; etc.) which would be considered as syntax errors in a standard Spectrum (thus causing error C, Nonsense in BASIC, to be displayed), are accepted by the shadow ROM (assuming that the syntax is correct, of course!) as valid commands. In other words, yes, you've guessed it, errors are VECTORED into the shadow ROM!

If, after all this seemingly interminable indirection, the command or program line is still not recognised, control is again vectored back into the old ROM and an appropriate error message is printed. Clever stuff. Even cleverer when one considers that the final error vector is in RAM, and can thus be altered. Here lies the crux! By the simple step of altering this RAM-based vector, one can cause all sorts of commands to be recognised as valid BASIC statements. Of course, efforts must be made to ensure that the new vector points to a routine that can handle the new syntax, and re-entry conditions for the ROM are satisfied on return, or all sorts of nasty things can happen. A vectored crash, although possibly impressive, is still as awful a consequence as any other sort of crash!

Remembering that the Spectrum BASIC ROM checks each line twice for syntax errors – once when the line is entered and then again at RUNtime – one must ensure that extended BASIC command routines can handle both error checks, and that the extended (ie new) command will be rejected by both the main ROM and by the shadow ROM. If this is not done then the final error vector will never be reached. This vector is held in the (new) system variable called. (with startling originality) VECTOR, and normally contains the value 01FC in usual Z80 fashion – that is, backwards! The address of this variable is 23735, and so its value can be determined by PEEK 23735 + 256 * PEEK 23736. This 'normal' address contained in VECTOR is the address to which control must jump (or even vector) to return to the error handling routine in the standard ROM.

So, to handle the syntax error checking, the extended BASIC routine must check its own syntax, ie that the statement is correctly formed, and that variables alluded to within the line are valid, and finally that the statement is terminated; either with a carriage return or with a colon as in multi-statement lines. As there is no point in duplicating routines, a user syntax analyser may as well take advantage of Sinclair's firmware and use the ROM routines for syntax analysis. Thus, RST 18 (which gets the character currently pointed to by the CH_ADD system variable) and RST 20, which gets the character following, will be required.

Now, as these two subroutines are in the main ROM, and control is currently in the shadow ROM's hands, a method is required which allow us to access these routines. It just so happens that the shadow ROM's RST 10 routine allows main ROM code to be called, by following the RST 10 instruction by two bytes defining the address of the main ROM routine. Confused? You will be! As a clarifying example, suppose the shadow ROM was currently paged in, and we needed to access the old ROM's RST 20 routine. Then, the code below will suffice:

```
RST 10        D7
DEFW 0020   2000
```

which is really very simple.

BEEP Extension

Now, enough should be known about the on-error vector system to actually extend the language – so an example is needed. Suppose the keyword BEEP was to be accepted with no parameters, so that on execution it would emit a short beep – something like the CTRL-G facility on the BBC Micro. Obviously, BEEP followed by pressing ENTER would normally cause the flashing '?' to appear, even with Interface 1 fitted, so that condition is satisfied. Next we have to decide the length and the pitch of the parameterless BEEP. Extensive use of a hearing aid has shown that Spectrum sound is most audible (to me) with a pitch of 10, and a suitable length would be about half a second. So, we're trying to make 'BEEP' by itself simulate 'BEEP 0.5,10' – a very worthwhile exercise! One problem here is that a duration of half a second is very hard to implement using this method, so to compromise use a value of one second.

First things first, then: A small routine is needed to fool the syntax-time check that a parameterless BEEP is a valid BASIC command. Here I must give appropriate acknowledgement to Dr. Ian Logan for the information gleaned from two of his books; 'The Complete Spectrum ROM Disassembly' and 'The Spectrum Microdrive Book', both published by Melbourne House. As a quick aside, it is interesting that Sunshine have also rush-released a Microdrive book by Andrew Pennell. It is not as useful as Logan's, and was written far too quickly. Anyway, back to the Spectrum . . .

If we adopt Logan's terminology, then listing 1 represents the syntax-time module – a very structured approach, and thus apparently preferable! It really does make a lot of sense to approach this task with an assembler and monitor available, and all my listings are ZX printed outputs from Hisoft's Devpač assembler/monitor package – very highly recommended.

As earlier stated, a fair number of main ROM routines will have to be called, and for reference, here is a list of the ones used in the syntax and runtime modules, their addresses, and their functions:

(Remember that expanded tokens such as BEEP are actually character represented; ie. BEEP is character 215).

GETCHR 0018 returns (in A) the code of the character currently pointed to in the current statement.

NXTCHR 0020 returns (in A) the code of the next character in the current statement.

STCK A 2D28 places the value held in A on the Spectrum's calculator stack.

BEEPCR 03F8 – using two values on the calculator (the pitch and the duration) this routine generates the requisite sound.

In addition to these main ROM routines, we will need ERROR6, ST_END and END1 from the shadow ROM. These are at 01F0, 05B7 and 05C1 respectively. ERROR6 returns to the normal error-handling routine, and will be jumped to if the command causing syntax errors is not BEEP: ST_END checks that the end of the current statement has been reached, and if so, causes a jump back to the interpreter if in syntax-time, and a continuation from the same place if in run-time. Finally, END1 is used as the final return point once every other operation has been undertaken. Of course, we must ensure that we change the contents of the VECTOR system variable before trying this new command out, or else nothing new will happen!

Listing 2 therefore constitutes the run-time module, and together listings 1 and 2 form the complete BEEP extension. Note that to use this properly, one must assemble the code, ensure that the shadow ROM's system variables exist (Logan suggests 'CAT' followed by <ENTER>), and alter the address contained in VECTOR. Naturally, more time and effort spent could easily result in more inventive and adventurous extensions to the Spectrum's BASIC. Before we venture into the world of the 6502 and the BBC again, one more example of vectored Spectrum extensions – this time one that EVERY Spectrum owner can try, as it does not rely on having Interface 1.

Spectrum 'BEEP' extension

```
GETCHR  EQU   24
NXTCHR  EQU   32
ERROR6  EQU   #1F0
ST_END  EQU   #5B7
BEEPCR  EQU   #3F8
STACKA  EQU   #2D26
END1    EQU   #5C1
BEEP    EQU   215
        ORG   64000

;SYNTAX MODULE

;LISTING ONE

        RST   16
        DEFW  GETCHR
        CP    BEEP
        JP    NZ,ERROR6
        RST   16
        DEFW  NXTCHR
        CALL  ST_END

;RUN-TIME MODULE

;LISTING TWO

        LD    A,1
        RST   16
        DEFW  STACKA
        LD    A,10
        RST   16
        DEFW  STACKA
        RST   16
        DEFW  BEEPCR
        JP    END1
```

Hex In BASIC

Here we take advantage of the DEF FN function to allow the use of hex (ie base 16) numbers in Spectrum BASIC. By treating every hex number as a string of up to four characters in length, it is very easy to write a short machine-code routine that will vector the defined function and return a numeric result corresponding to the hex string. For completeness and security, there are also comprehensive error checks, so that bad hex (ie if the string contains characters that don't make sense as hex digits) and numbers over 65535 (in fact, over 4 hex digits) will be rejected. At the same time, both lower and upper case digits will be accepted – at the same time, in the same string. The function can be called in two ways, either with a quoted string constant:

PRINT FN H("03E8")

or with a string variable:

PRINT FN H(a$)

So, how does this short little program work? Well, whenever a user-defined function is invoked (ie by using the FN call), the system variable DEFADD contains the address of that particular function information. Four bytes above this address is the address of the argument (for string DEF FN's), and the byte immediately following this contains the length of the string. Therefore, if the contents of DEFADD (which is at address 23563) are put into a register pair such as HL, this vector can be used (yet again!) to point to the relevant information. So, the first few bytes of this little routine fetch the argument's address, increment it by four, load the DE register pair with the next two bytes (the address of the string) and finally load the A register with the string's length. This length is then compared with 0 (by an AND A instruction, which has the same effect as CP 0 but is one byte shorter and 4 T states faster). If the length of the string is zero, then the program jumps to ERRORA, which by using the Spectrum's RST 8 error reporting system, stops the program with BASIC's error A – invalid argument – report. The length is then compared with 5, as we can only handle 4 digits (clever people could easily modify this), and if the length is 5 or greater, error 6 – Number too big – is generated.

As the intention is to use HL as a two byte accumulator, this is now cleared (LD HL,0), and B is loaded with one less than the string length. If B contains zero, indicating a string of only one character, then ONECHR is jumped to, which as its name suggests handles the final character in the string. Otherwise, a loop is entered (controlled via a DJNZ instruction using the value that we have just given the B register) that checks each character for validity (that is, it must either be a digit between 0 and 9, or a letter between A and F or a and f). If none of these conditions is satisfied, then error C – nonsense in BASIC – is generated. The value (from 0 to 15 decimal) of each digit is then added to the current sum (contained in HL) and the whole multiplied by 16, as each digit is 16 times more significant than the next (hence hexadecimal!).

Hex string defined function

```
DEFADD  EQU   23563
        ORG   65000
        LD    HL,(DEFADD)
        INC   HL
        INC   HL
        INC   HL
        INC   HL
        LD    E,(HL)
        INC   HL
        LD    D,(HL)
        INC   HL
        LD    A,(HL)
        AND   A
        JR    Z,ERRORA
        CP    5
        JR    NC,ERROR6
        LD    HL,0
        LD    B,A
        DEC   B
        JR    Z,ONECHR
NIBBLE  LD    A,(DE)
        CP    #60
        JR    C,CAPLET
        CP    #67
        JR    NC,ERRORC
        SUB   32
CAPLET  SUB   48
        JR    C,ERRORC
        CP    10
        JR    C,DDIGIT
        SUB   7
        CP    16
        JR    NC,ERRORC
DDIGIT  OR    L
        LD    L,A
        ADD   HL,HL
        ADD   HL,HL
        ADD   HL,HL
        ADD   HL,HL
        INC   DE
        DJNZ  NIBBLE
ONECHR  LD    A,(DE)
        CP    #60
        JR    C,LETCAP
        CP    #67
        JR    NC,ERRORC
        SUB   32
LETCAP  SUB   48
        JR    C,ERRORC
        CP    10
        JR    C,DIGITC
        SUB   7
        CP    16
        JR    NC,ERRORC
DIGITC  ADD   A,L
        LD    L,A
        LD    A,0
        ADC   A,H
        LD    B,A
        LD    C,L
        RET
ERRORC  RST   8
        DEFB  11
ERROR6  RST   8
        DEFB  5
ERRORA  RST   8
        DEFB  9

 1 DEF FN H(H$)=USR 65000
10 INPUT A$
20 PRINT FN H(A$). GO TO 10


PRINT FN H("FFFF")

65535

LET A$="00C9"

PRINT FN H(A$)

201
```

The end of this loop is reached when one character is left, and the routine starting at ONECHR handles this by adding it (after error checking) to the contents of HL, but returning the result in BC so that it can be retrieved by Spectrum BASIC after the RET instruction. Thus, Hex is easily added to Spectrum BASIC with the minimum of fuss.

BBC Interrupts

Now on to the BBC Micro. BBC BASIC is extremely comprehensive and doesn't really need extending, which is rather lucky as it's a fairly awkward process! (Try using OS 1.2 *LINE and *CODE if you want). However, as I said last month, vectors are used for 6502 interrupt handling, and there is absolutely no reason why we shouldn't alter these vectors to useful ends. Thus, we can make interrupting devices cause all sorts of strange and wonderful effects. Naturally, care has to be taken that all the normal consequences of interrupts are taken care of, otherwise a Spectrum-style crash may occur! Under normal circumstances, the parallel printer port, the user port, the Tube, the disc drives and the keyboard are serviced by interrupts.

The User Guide gives dire warnings not to use NMI's if a DFS ROM is fitted, as these are handled specifically by the DFS, so we'll ignore that particular sort of interrupt. This leaves BRK and IRQ type interrupts, which the operating system kindly vectors through &202 (BRKV), &204 (IRQV1) and &206 (IRQV2), depending on the type of interrupt. This is where the vector is 'grabbed' and the interrupt redirected.

One use of the BRK instruction is to alter the message printed (so that you can make the computer swear at you if that's what you really want – I think not). So, BRK vectoring can also be discounted for the moment. This leaves just IRQ vectors to play around with.

IRQV1 and IRQV2 represent two effective levels of IRQ (Interrupt Request) trapping, as described in the User Guide (page 466), although the amount of information given here is desultory. What actually happens is that on an IRQ the operating system indirects through IRQV1 to a special service routine that deciphers the source of the interrupt. If this routine cannot handle the interrupt then indirection through IRQV2 occurs. Such peripherals as the user 6522 cause this sort of situation, and therefore hardware can be controlled this way. This is really beyond the realms of this article, as we are trying to demonstrate vectoring to ALL BBC owners, so specialist hardware applications will not be discussed. Acorn provide (that is, sell!) an application note for people that want to venture into this area.

So it is obvious that what we want to do is intercept an IRQV1 vector, re-direct it through our own routine, and then return to the operating system through the original contents of IRQV1. This value is (on my BBC with OS 1.2, Acorn DFS and a million sideways ROMS) normally &DC93, found by PRINT " (?&204 + 256 * ?&205). Therefore, we have to change the contents of these two locations to point to our short servicing routine, and then return by a jump to &DC93. A few points here: our interrupt routine must be appreciably shorter than the time between interrupts (obvious), we must not enable interrupts during the routine (otherwise the stack will fill up rather fast!), and it is as well to preserve the current contents of every register. The accumulator is saved in zero-page location &FC by the operating system, so that's OK, but X and Y (the status register is automatically saved) will need saving on the stack, as unlike the Z80, the 6502 does not have an alternate register set. A suitable start and end for our extra routine might be:

```
TXA                 PLA
PHA                 TAY
TYA                 PLA
PHA                 TAX
(our routine . . .) JMP &DC93
```

Having now worked out the mechanics and the necessary precautions for IRQ handling, we next need an application. The example given for the Spectrum, that of a continuous 24 hour clock display, is redundant on the BBC as this computer already has a clock. In fact, because of the sheer completeness of the BBC system, it is very difficult to come up with a serious applications for vectored interrupts.

To avoid all sorts of nasty mis-vectorings, the contents of IRQV1 must only be altered with the interrupts disabled, and this therefore precludes an alteration from within BASIC. So, as a demonstration of the technique, the short program below alters IRQV1 with interrupts disabled, and at the same time generates a small routine that causes a beep every interrupt – effectively continuous sound.

Lines 210 and 220 are the actual additional routine, and further examples can be shown simply by replacing the code here. For instance, changing line 210 to read LDA #ASC "A" will cause the character 'A' to be printed on every interrupt. Again, not too useful an application, but it does serve to demonstrate.

Well, by now vectoring and indirection should be that much clearer to you all, and especially their use in processor interrupts. For those who have computers with processors not covered in this short series, such as the Dragon 32 with its 6809, have no fear – the basic concepts still apply, 6809 code is not so far removed from the 6502's, and readers should find little problem in conversion of applicable routines.

Fig 3. BBC continuous sound

```
 10 DIM P% 100
 20 PROCsetup
 30 CALL BEGIN
 40 END
 50 DEFPROCsetup
 60 FOR C=0 TO 2 STEP 2
 70 [OPT C
 80 .BEGIN
 90 SEI
100 LDA #START MOD 256
110 STA &204
120 LDA #START DIV 256
130 STA &205
140 CLI
150 RTS
160 .START
170 TXA
180 PHA
190 TYA
200 PHA
210 LDA #7
220 JSR &FFEE / OSWRCH
230 PLA
240 TAY
250 PLA
260 TAX
270 JMP &DC93
280 ]
290 NEXT C
300 ENDPROC
```

E&CM

BLANK CASSETTES!

TOP QUALITY PROFESSIONAL BRAND COMPUTER/AUDIO CASSETTES AT BUDGET PRICES

Packed in boxes of 10 cassettes. Complete with labels, inlay cards and library cases.

Prices include VAT post & packing

Length	Box Price (10)	Qty.	Amount
5 mins	£4.35		
10 mins	£4.40		
12 mins	£4.45		
15 mins	£4.50		
30 mins	£4.70		
60 mins	£5.30		
90 mins	£7.00		

Cheque/Postal Order enclosed for £

NAME ....

ADDRESS ....

....

PROFESSIONAL MAGNETICS LTD.

Cassette House, 329 Hunslet Road, Leeds Tel: (0532) 706066

TRADE ENQUIRIES WELCOME

ECM01

REAL-TIME PLUG-INS for your DRAGON 32

Use your DRAGON 32 to measure voltages – temperatures and switch devices on and off.

ADC-09 Fast analog to digital converter. Supplied with some suggested applications. Plugs into cartridge socket **£29.50**

DR8CHR Relay output box .... **from £16.75**
Up to eight relays possible per box. Supplied with operating software.

D232 RS232C Interface **£35.00**
A full spec. RS232C interface with PROGRAMMABLE Baud rate selection. Supplied with operating software for listing and for accessing data bases.

Other plug-ins and ideas for the DRAGON 32 please send Large SAE for full catalogue to:

PNP Communications

62 Lawes Avenue, NEWHAVEN, East Sussex BN9 9SB
Telephone: (0273) 514465

Callers by appointment only please.

ECM01

M & J SOFTWARE

DRAGON fig-FORTH **£12**
A cassette-based implementation of FORTH which includes a powerful text editor and a 6809 macroassembler. All the power of Basic is retained by being able to access Basic commands from FORTH. Do not be fooled by the low price of this package – it represents unbeatable value for money and comes complete with extensive documentation.

fig-FORTH ASSEMBLY SOURCE LISTINGS .... **£7 each**
Available for the following processors:-
6502, Z80, 6809, 8080, 1802, 9900, 6800, 68000, 8086/88, & PDP11

MVP-FORTH ASSEMBLY LISTINGS .... **£7 each**
Available for 6502, 8086/88 & 8080 processors.
These listings provide the source for an implementation of FORTH up to 79 standard.

fig-FORTH INSTALLATION MANUAL **£5**
A complete 'how to do it' guide to the implementation of FORTH from the above listings. This manual contains the FORTH source written in FORTH, an editor, an extensive glossary and lots more.

ALL ABOUT FORTH by Haydon **£7.95**
An excellent reference book with cross references to fig-FORTH, the FORTH-79 standard and Starting FORTH. This book should be next to every FORTH programmer's computer.

6809 & 6502 MACROASSEMBLERS .... **£5 each**
Written in fig-FORTH, these listings require the minimum of alteration for any fig implementation. Copies on tape can be supplied for Dragon and Microtan users at £1 extra.

DRAGON COMPANION **£4.95**
A useful source of information on the workings of the DRAGON. Includes a full 6809 disassembler.

Cheques & P.O's to:-

M & J SOFTWARE 34 Grays Close, Scholar Green, Stoke-On-Trent, ST7 3LU
Telephone: (0782) 517876

ECM01

INSTRUMENTATION AND CONTROL FOR THE BBC

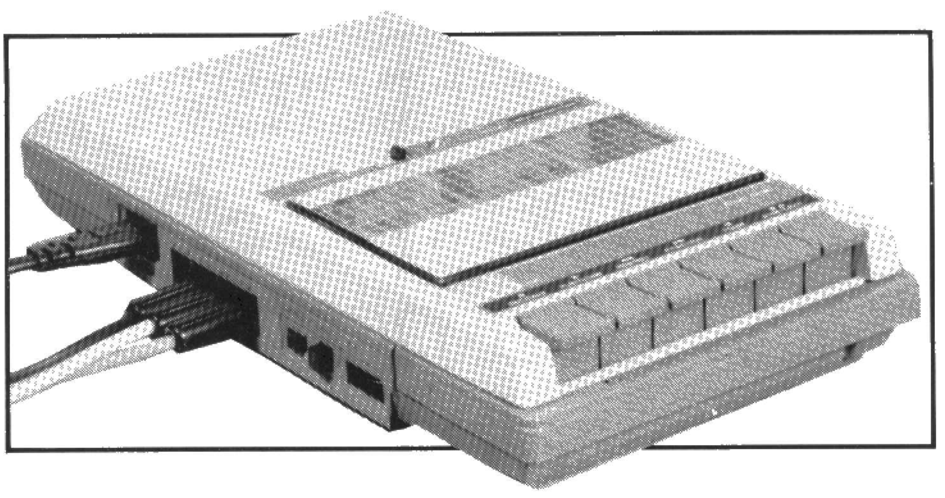
1. **ANALOGUE DISTRIBUTION UNIT.** Will accept up to four analogue sensors and three switching devices, or two switching and one low speed Digital to Analogue converter, or light pen.
2. **DIGITAL DISTRIBUTION UNIT.** Will accept up to eight switching devices or sensors that give digital output, with or without handshake.
3. **SENSORS** available for temperature, light, pH, and liquid flow. Others under development.
4. **CONTROL DEVICES** available are, relay contacts and opto-isolated mains socket. Other control devices under development and specials to Order.
5. **SOFTWARE** to exercise individual sensors or control devices with enough remarks to enable you to modify it to suit your requirements.

For further details write or phone:

DEDICATED MICROPROCESSORS LTD.

Regeneration House, School Road, North Acton, London NW10 6TD
Tel: 01-965-2841

SANYO DR101



Robert Penfold has used Sanyo's new data recorder with a wide variety of popular micros. His tests reveal that the unit is capable of performing reliable LOAD and SAVE operations.

Of all aspects of home computing, perhaps the one that gives rise to the most complaints and the most heartache is the storage of programs and data on cassette tapes. In many cases, rude remarks about equipment are unjustified, and the cause of the problem lies in such things as the failure to regularly clean tape heads, or in the unwise use of 'three-way' mains adaptors, which are a prolific source of electrical "pops" and "crackles", instead of proper distribution boards. However, to record data at speeds of 1200 baud plus is asking a lot of domestic cassette recorders.

There are now appearing on the market special 'computer-compatible' cassette recorders, designed to give improved performance in this application. The first of these to become widely available was the Sanyo DR101.

Externally, this does not look very different from other cassette recorders. It has a tape counter, the usual piano-key controls, and a condenser microphone. The special features are to be found in two extra switches on the side.

Something Extra

The first of these is a three-position slider. The first position gives 'normal' operation. In this mode, the recorder records from the built-in or an external microphone (or other source) with automatic recording level; and playback volume is controlled by the conventional rotary volume control.

The other two settings are the 'DATA' settings for computer operation. The middle position, labelled 'MONI-ON', allows the sound of data saving and loading to be heard at subdued volume through the speaker. The last setting, labelled 'MONI-OFF', mutes the speaker. This is the only difference between them.

The record level is set automatically in the data and normal modes. In addition, in the data mode the playback level is fixed, the volume control having no effect at all. This gave rise to some initial misgivings, as volume settings can be critical when loading programs.

The second extra switch is a push-switch, which gives normal phase in the 'out' position, and reversed phase when 'in'.

Cue and review facilities are available on FFWD and REW. The FFWD and REW controls bypass the REM socket, so it is not necessary to pull out the REM plug, or turn on the motor from the computer, in order to position the tape. Full auto-stop would have been useful in conjunction with this facility, but the auto-stop only operates during record and playback. The pause control is mechanical, not just a switch type retracting the pinchwheel, but leaving the motor running and electronics active for a clean restart. There is a two-colour indicator LED, red for record, green for everything else. The tape counter is accurate and easy to read. Operation is from AC mains, or 4 C size/U11 batteries, and there is also a socket for an external 6V DC power source.

LOADed Results

We have been able to test the DR101 with several popular home computers. To take these in alphabetical order, with the BBC (Model B) we have used it extensively both for programs and for data (Wordwise text) files. At the 300 Baud rate, it can be used with the phase switch in either position. At the (standard) 1200 Baud rate, it requires the phase to be reversed on its own recordings, but, strangely, with the BBC 'Welcome' cassette, it will only load in normal phase. No problems have been experienced LOADing programs, but with the Wordwise tapes, occasional failures have been experienced. This seems to be due to the playback volume on the DATA settings being on the high side for the BBC computer. It has always been possible to load these tapes by switching to NORMAL mode, and using a moderate volume setting. When loading tapes made on other recorders, it is also sometimes necessary to use NORMAL mode.

The Dragon-32, despite the lack of a tone leader to stabilise an automatic recording level circuit, has the reputation of having a good cassette interface. With the DR101 it is possible to load both programs and data with the phase switch in either position. No problems at all were experienced loading the DR101's own recordings, but again, when loading tapes made with another recorder it was occasionally necessary to switch to the NORMAL setting.

The ORIC-1 has earned a poor reputation for cassette loading, but the manufacturers have always stoutly defended the computer's hardware. We have always found the slow rate to be successful, but the standard, 2400 Baud rate to be unusable without additional external processing. However, with the DR101, both speeds can be used successfully, without add-ons. The slow rate (300 Baud) works with the phase switch in either position, but on the faster rate the situation is rather like the BBC. The phase must be reversed for the DR101's own recordings, but the Oric 'Welcome' cassette requires normal phase. With this recorder the Oric's motor control relay has always worked correctly. With some other (older) cassette recorders, it is inclined to stick on, presumably due to them drawing a higher current.

The DR101 was also used with both the Sinclair ZX81 and ZX Spectrum. With both these computers it is possible to load with the phase switch in either position. A further advantage for Sinclair owners is that the DR101 appears to have internal switching so that there can be no feedback to the computer during recording. This means that it is not necessary to pull the earphone plug out during SAVE operations, or to disconnect the MIC lead when loading. This is no small point. The ease with which the DR101 can be used with the two Sinclair machines tends to suggest that it was designed primarily for use with these.

Normal Use

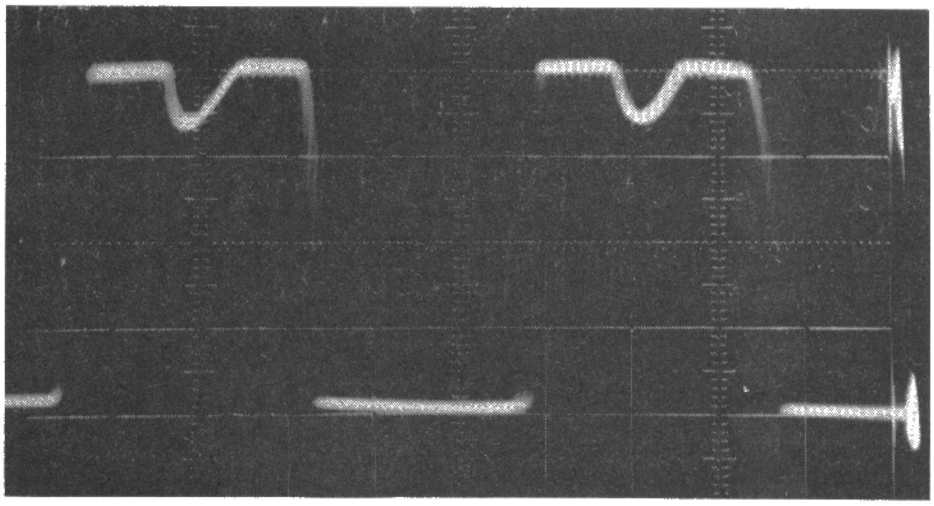
In normal operation, the DR101 is a typical modern cassette recorder of above-average quality. The motor is smooth and quiet, and recordings made with the built-in microphone have a commendably low level of background noise from this source. In fact, noise of all sorts on recordings and playback is well subdued. The erase head completely obliterates old recordings – this is often a weak point of cassette systems.

The sound quality from the small internal speaker is as good as could reasonably be expected. Used with an external speaker of good quality, the quality of sound is much improved, though a shade too treble for my taste. There is no tone control.

Technicalities

Signals are recorded normally whether the normal or the data mode is selected. On playback in the data mode the signal is subjected to a lowpass filter followed by a modest amount of amplification. This amplification is sufficient to clip a strong signal, such as that obtained from a computer. It was possible to mimic the internal processing using a two pole 1 kHz highpass filter followed by an amplifier having a voltage gain of about 3 times.

If a 1 kHz sinewave signal is recorded and played back in the data mode the result is a reasonably good quality sinewave. With a squarewave signal (the usual output waveform of a computer) the basic output of the recorder suffers severe waveform distortion due to the usual cassette recorder frequency response limitations and phase shifts. The processed signal is much closer to the original squarewave input, but there is a slight "notch" in the lower part of the waveform. This moves to the upper section of the waveform when the phase reversal switch is operated, and also seems to become slightly more pronounced.


DR101 output waveform (Data mode).

While the processed output signal is not perfect, it is closer to the original than the output of a normal cassette recorder, and many computers seem to find this signal more acceptable than an unprocessed signal.

Conclusions

The DR101 was designed to be the ideal machine for use with home-micros, and in practice it seems to live up to expectations. It is somewhat more expensive than an ordinary cassette recorder having similar facilities (but without the data recording facilities), but it seems to give reliable results with a wide range of computers having very different cassette interfaces. A further advantage is that it is generally more convenient in use. For anyone having cassette loading or saving problems it would seem the obvious choice, and might be worthy of consideration just for its greater convenience in use.

Having used this machine extensively with a number of home-computers it is difficult to level any real criticisms at it. So far it has never failed to load a program, with one or other of its modes giving satisfactory results.

E&CM

Free Readers Advertisements

Buy — Swap — or Sell

- *Want to Sell?*
- *Want to Buy?*
- *Want to Swap?*

Then Readers Exchange is the answer

Next month we welcome a new service to the pages of *E&CM*. It is aimed at readers who want to buy, swap or sell computers, peripherals, components, books... in fact just about anything. The only stipulation is that all adverts must be from private individuals — no trade advertising.

To make use of this new service, print clearly on a plain sheet of paper the details of your advert together with your name, address and, if applicable, your telephone number (we are unable to offer a box number service). Send this to:

Readers Exchange
Electronics and Computing Monthly
155 Farringdon Road
London EC1R 3AD

Your advert will appear in the next available issue of *Electronics & Computing Monthly* and those of you who are quick off the mark could find it included in our February issue.

LETTERS TO THE EDITOR

Send your programming or construction hints, comments, queries, or complaints! to *The Editor, E&CM, Scriptor Court, 155 Farringdon Road, London EC1R 3AD.* £10 is paid for the star letter of the month.

★ £10 Star Letter

Spectrum tape backup debugged

Sir,

I was very interested in your "48K Spectrum Tape Backup" program by Tubb Research in your December 1983 issue.

Unfortunately after typing the program in and debugging it, I found that the program would not always load programs, especially programs of any length.

After carefully studying the machine-code routine for loading of the main program I found that nowhere was the length of the program being put into the DE register. Hence this is why the program would not work properly. I have included the amended machine code loading routine as well as the corrected program lines for you to print so that other readers may be able to alter their programs to work properly.

I also found that more than 32K of memory was needed for some programs so I have also included some amended lines so that about 36500 bytes of memory can be copied.

I hope this material will be useful to readers as this program enables back-up copies to be made very easily.

Amended machine code loading routine:

```
LD A,255
LD D,19
SCF
LDIX,32 768+40
LD DE,0000
CALL 0556H     length of program
               filled in at lines
               461-463
RET
```

BASIC program changes:

```
170 LET byte=32: LET add=65280:
      LET res=890
461 LET k=INT(length/256)
462 POKE 65305, length-(256*k)
463 POKE 65306,k
920 DATA 62,255,22,19,55,221,33,
      40,128,17,0,0,205,86,5
```

To allow programs of about 36500 in length make following changes:

```
 40 CLEAR 28449
260 LET a=28450
540 POKE 65298,74
550 POKE 65299,111
890 DATA 17,17,0,175,55,221,33,34,
      111,205,86
920 DATA 62,255,22,19,55,221,33,
      74,111,17,0,0,205,86,5
```

T. M. Lewis
Bangor, Gwynedd

EDITOR'S NOTE

We would like to apologise to readers for the mistakes in the Spectrum back-up listing published last month. While the program we received worked, the listing was incorrect. The author has sent in the following amendments to the program which have been checked. Readers will see that a similar approach has been used to that suggested by T. M. Lewis:

```
170 LET byte=31: LET add=65280:
      LET res=890
461 LET k=INT(length/256)
462 POKE (program+4),
      Length-(256*k)
463 POKE (program+5),k
920 DATA 62,255,55,17,0,0,221,33,
      40,128,205,2,8
```

Remove line 565

Some lines of the program proved difficult to read when the listing was first published, and to assist readers further these are reproduced below:

```
 90 LET byte=56: LET add=65000:
      LET res=820
170 LET byte=29: LET add=65280:
      LET RES=890
570 POKE 65301,length-(256*k)
580 POKE 65302,k
```

BBC EPROM Programmer – The authors write

Sir,

We would like to make the following comments after feedback from some of the several hundred people who built our "EPROM Programmer for the BBC Micro" published in the Oct/Nov 1983 issues of *E&CM.*

1) Some EPROMs eg. Mitsubishi need 21 volts applied to pin 1 to write and 5 volts to read. All our testing has been with Hitachi and Intel types which appear to have an internal resistor to produce 5 volts in the absence of the 21 volt programming supply. If you wish to be able to program all EPROMs, a solution is to solder a diode (a 1N4148 will do) between pin 1 and pin 28 of the ZIF (cathode to pin 1) and a 100k resistor between pin 1 and 0 volts. These can be neatly added to the underside of the board. Be warned! The BBC Micro does not have 5 volts on pin 1 so these EPROMs will not necessarily work in it, although they will work on some paged ROM extension boards which have this pin tied to 5 volts.

2) A small number of constructors have found that the programmer sometimes ceases to program part way through an EPROM. This has been traced to noise spikes on the local mains supply, which are large enough to operate the reset line. Cures have been effected in several ways, including the addition of a 2K7 resistor on the reset line (pad 14 of the cable connection) to 5 volts, using a regulated mains supply to both micro and programmer and, finally, obtaining the programmer 5 volts from the BBC micro whilst using a battery for the 21 volt supply. Unfortunately, no one action cures everybodies problem so individual attention is necessary.

3) The program listing as published was produced from a working program but suffered slightly during magazine production. The corrected lines are given below.

Program 1

```
Line 890 STA eprom address low
     900 LDA #&3F
     970 .STA IC4 control 1
```

Program 2

```
Line 1890 PRINT"File name = ";N$''
              "Load address = &";L%
     3670 PRINTTAB(M%)
              "1) Define as an ASCII
              character".
```

We hope this information will be of use to your readers,

Peter Simpson and Brian Alderwick

ZX memory expansion

Sir,

Your reader B. Prajzner, (Letters December '83) obtained the same value for P_RAMT as I did after I fitted an upgrade kit for the Issue 2 Spectrum.

IC26 was a 74LS157 and the memory chips, ICs 15-22, were TMS4532s. An input of IC26 (pin 10) is not connected anywhere on the board, but is brought out to a pad between IC3 and the ULA. I linked this pad to the pad marked '0V' and I now own a fully grown up Spectrum.

I. D. Turnbull
Stanley, Co. Durham

Sir,

With reference to B. Prajzner's problem (Letters December '83), I have experienced getting odd values for RAMTOP with an expansion of the ZX81's memory.

After a lot of chip-swapping, I realised that the problem was in one IC socket, with some of the IC pins not getting the signals properly. The solution is to solder short flying leads direct to the IC pins. The way I detected this was very crude: I used a digital voltmeter to read the voltage on the IC pins which should have been connected parallel (eg. the address pins). The meter gives a reading between 0 and 5 volts, presumably reflecting the time-average of the 0's and 1's on the pin. I found that voltages read off one IC were different from those on the corresponding pins of the other ICs – hence the soldering. Of course, this will only show up a problem on pins which are paralleled between ICs, but it is a lot easier than trying to analyse signals with proper instrumentation.

A. C. Walkland
Southampton

BOOK AND SOFTWARE REVIEWS

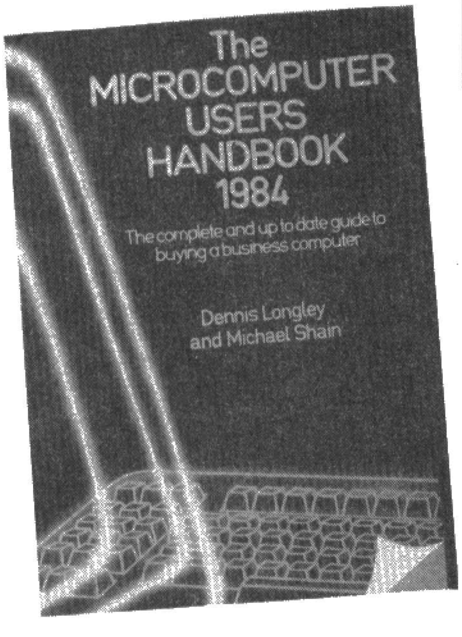
Each month Harry Fairhead will look at a selection of recently published books and software.

The Microcomputer Users Handbook 1984
by Dennis Longley and Michael Shain
Macmillan, 1983.

This microcomputer annual is a large format softback with 377 pages illustrated throughout with appropriate black and white photographs and helpful line drawings. In the computer industry as a whole the idea of yearly handbooks is nothing new. Most large computer installations have a standing order for one or two yearbooks and within their typically huge budgets such small regular purchases are hardly noticed. A microcomputer handbook would seem like a fairly obvious addition to the range already available and indeed the Microcomputer Users Handbook 1984 is a very traditional offering. It includes not only the advertising that characterises its established counterparts but also a reply paid card committing you to receiving the handbook in subsequent years!

The book starts off with a chapter on "How to buy a microcomputer". This is oriented towards the business user and covers familiar ground. The next chapter explores how the different parts of a microcomputer system work. Chapters three and four deal with actually using the micro. In my opinion Chapter five dealing with current trends in microcomputing is the most interesting. It gives an account of local area networks, the use of micros in conjunction with "the corporate mainframe", telesoftware and videotex, and reviews some currently popular operating systems. The last half of the book is comprised of a microcomputer and peripherals survey. The surveys present a factual rather than a critical appraisal of a very large and comprehensive collection of hardware. This would be useful to anyone constructing a list of machines that satisfy some criterion. Although the handbook is directed at business users, a number of machines usually thought of as home computers (including the Spectrum and ZX81) are included.

The Microcomputer Users Handbook is full of interesting information. It is well written and well produced but I'm not entirely sure at whom it is aimed. Large computer installations that also have to look after a growing number of micros will probably place a standing order but even in the fast moving field of microcomputing I'm not at all sure that I would want a new copy in just 12 months time!


The MICROCOMPUTER USERS HANDBOOK 1984
The complete and up to date guide to buying a business computer
Dennis Longley and Michael Shain

SELECTED SOFTWARE

Dragon Composer
for the Dragon 32
Microdeal

The Dragon doesn't have a special sound effects generator but instead has a completely general six bit D/A converter (DAC) which can be programmed to produce virtually any conceivable sound. To create a particular sound all you have to do is send a stream of numbers corresponding to the waveform to the DAC. The problem is that this has to be done at such a speed that 6809 assembler is the only suitable language for the task. Of course this assembler is by no means a convenient language to use and this is where Dragon Composer comes in. Essentially Dragon Composer is a machine code routine that will take a list of numbers stored in memory and transfer them to the DAC. To get the list of numbers into the correct format a BASIC program is provided that will 'compile' user-supplied DATA statements and store the result in memory. In fact the Dragon Composer is a little more sophisticated than this description suggests in that it has four independent voices each one with its own sound quality. This means that if you construct the correct DATA statements you can make the Dragon play up to four part harmonies.

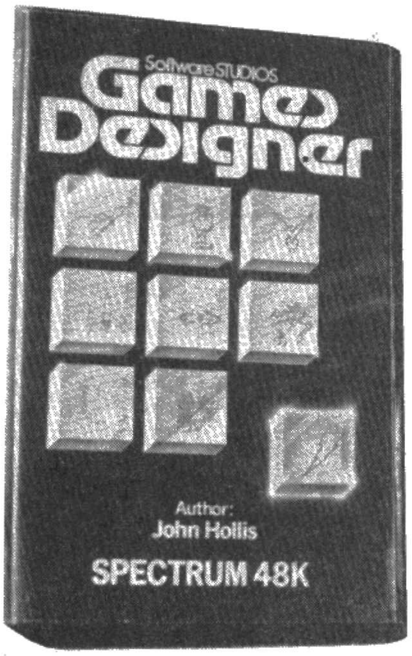
All this sounds good in theory and indeed the Dragon Composer tape includes enough examples to indicate that it is possible to program some complicated and effective music but is restricted to fairly short pieces of music with a limit of 720 notes – usually less than five minutes worth). However, it is important to be aware of the difficulties in using this software. Constructing the correct DATA statements to produce the notes of the required pitch and duration is very similar to using the Dragon's PLAY command and so presents the same level of difficulty. Writing music as a long list of one letter note names and duration values is not the best way to compose new music and is only just acceptable as a way of transcribing written music to computer. Another factor that makes Dragon Composer even less suitable as an aid to composition is that the translation of the DATA statements to the numbers necessary to control the machine code program takes about 1 second per note! For a piece of music of any size this time adds up to quite a wait until you can hear your masterpiece. Of course if the composition is a little wrong then it is necessary to go through the whole cycle again to correct it! A final problem is that there appears to be no way of controlling the volume of an individual note, and so most of the music starts to sound a little mechanical (!).

Dragon composer is suitable for transcribing existing written music to computer and producing it in a form that can be incorporated into other BASIC programs. Using it you can produce very exciting four part harmonies but it is a lot of work. What Dragon Composer is less suitable for is composing new music – you need an easier way of experimenting and quicker feedback about what things sound like – but it does provide your very own performing quartet.

Games Designer for the
48K ZX Spectrum
Software Studios – marketed by
Quicksilva

Games Designer is a remarkable program in that in one general package it offers you the opportunity of playing all of the 'zap the aliens' type games that you could want! Writing a program that allows the user to construct such a wide range of games sounds like an impossibility. However, the fact that it has been done shows just how limited the range of 'zap the alien' games really is! Games Designer is a machine code program that allows you to define the shape and movement pattern of 32 sprites each 12 by 12 pixels. (If you want to know about sprites see Parts 1 to 3 of Mike James series on "Micro Graphics Techniques"). Instead of adding a completely general sprite capability to ZX-BASIC, which would still leave you with the problem of writing quite a lot of code to define the rules of a game, the sprites are given special properties. For example sprites 0 to 15 are designated 'aliens' and sprites 24 and 25 are 'missiles' etc. By defining a sequence of sprite shapes, internal sprite animation can be produced automatically. The sprite animation is very smooth and convincing, as are all of the features of Games Designer.

Defining new sprites is only one level, and perhaps a very trivial level, at which a game can be altered – space invaders with different shaped invaders and missiles is still space invaders! Games Designer offers four games formats – invaders type, asteroids, scramble, and berserk. (If you know the games of the same name then you will understand these descriptions). Once you have selected the type of game to be played then the modifications that can be made seem extensive but it is impossible to get away from the underlying game format. New



sprites for the aliens, missile bases, missiles etc., can be defined as can movement patterns, backgrounds, sound effects, scoring, attack patterns etc. All of these changes are sufficient to produce games that look different and do give the user the feeling of creating their own games, but you are working within a games world that has been strictly defined by the author. This is not a criticism of Games Designer, which is an excellent piece of well written software, but a statement of how limited such games are.

There are games that I can think of that cannot be produced using this program but they are not strictly speaking of the 'zap the alien' type. If you want to write games that go outside the standard formats then Games Designer will disappoint you. However, if you enjoy conventional computer games then Games Designer supplies all of them in one program! This package also has an educational application in encouraging games addicts to consider some of the problems involved in programming their favourites. If however they become sufficiently keen on programming to learn a high level language such as BASIC they may be distressed to discover that it doesn't offer the same facilities to manipulate sprites as those found in Games Designer. Finally it is worth mentioning that the program comes complete with eight different programmed games which on their own make it very good value for money!

The vast array of books and software currently available presents the micro-computer user with a bewildering choice. This page is intended to help readers by presenting an informed and considered opinion about a couple of books and pieces of software each month. Because space is limited, this review page will be selective, and only books that seem to offer a good deal will be included. Every book reviewed on this page can be obtained through the ECM Book Service.

Computer Images
State of the Art – Joseph Deken

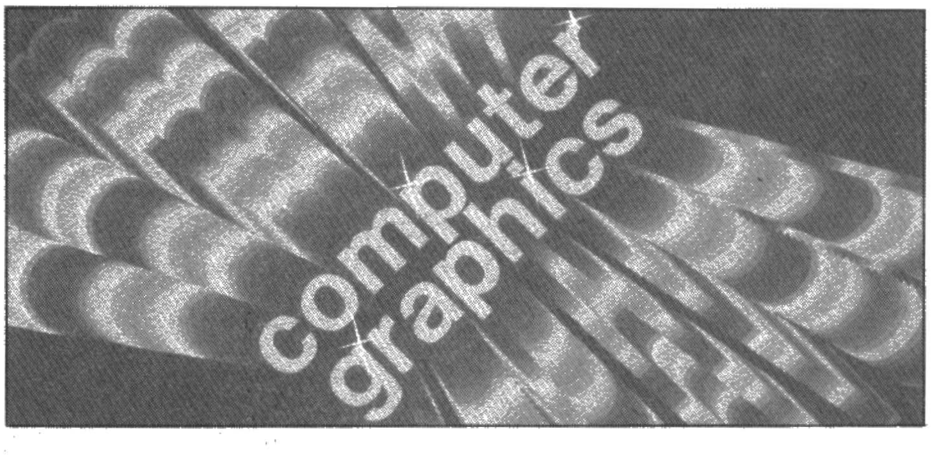
All the images involved in this superbly produced book are those created by specialist machines, revealing a depth and quality that would be difficult to achieve from a home computer. The variety and dissimilarity of such pictures make it a very attractive book just to have around, notwithstanding some intelligent and precise textual backup from Deken.

The author seems anxious to deflect any criticism that the creation of such graphics is somehow detached from man and strives to emphasise their usefulness. Primarily he stresses the role of the computer as an artistic tool, a sophisticated set of brushes and paints, whereby works of art can be created more precisely and distinctly. He claims it thus as 'an extension of the brain'. He also cites the uses of graphic techniques in areas like architecture, where 3D visuals can cut down the time factor required for the making of scale models; medicine, where thermographic techniques can quickly detect cancer growth and flight simulation, an area where games enthusiasts would appreciate the realistic effects created by the minuteness of the pixels used by the 'realicorders'.

Some of the most striking images are represented in the Fantasy section. Here he concentrates upon the purely aesthetic side of the 'art' where such technicolour patterns and landscapes lead him to rather whimsically remark 'they are as alluring as the siren's song'. His enthusiasm may be a little overboard but it is difficult to believe that such images are really the results of complicated mathematical formulae.

Deken claims that 'there are few limitations to the productive potential of the computer'. The quality and variety featured in this highly colourful book justify this claim. The first-class reproduction and plentiful illustration make it an extremely attractive addition to the coffee table and given the season, would make a rather nice Christmas gift.

Liz Gregory



"THERE MUST BE A COMPUTER DEALER I CAN TURN TO FOR GUIDANCE...."

If you're still staggering through the computer jungle and not getting sensible answers to your questions, we have some good news:
Now you can turn to **professional** people who are capable of giving you sound advice on practically all aspects of popular computing.
They all have one thing in common:
They are **COMPUTERS FOR ALL** dealers.
A Computers for All dealer is different from the normal Computer retailer.

Not surprisingly he won't try to sell you things like cameras or cosmetics; stationery or sealing wax.
He will, however, be capable of answering sensibly almost any question you have on computers and computing, and have readily available a wide range of popular computers, hardware, software, books, and peripherals.
So why not call in at your local **COMPUTERS FOR ALL** dealer today?
He can lead you in the direction you want to travel.

The shops where people matter:

AVON
MERCATOR COMPUTER SYSTEMS
3 Whiteladies Road, Clifton, Bristol. 0272 731079.

BERKSHIRE
KENNETH WARD COMPUTERS
Verve House, London Road, Sunningdale. 0990 25025

CHANNEL ISLANDS
MEGA LTD.
7 Anley Street, St. Helier, Jersey. 0534 72263

MICROTEL
Camp-Du-Roi Vale
Guernsey. C.I
0481 53436

CHESHIRE
THE COMPUTER CENTRE
68 Chestergate, Macclesfield.
0625 618827

CORNWALL
COMPUTAVISION
4 Market St., St. Austell
Cornwall. 0726 5297

FAL-SOFT COMPUTERS
8 St. George's Arcade
Falmouth TR11 3DH. 0326 314663

DEVON
A & D COMPUTERS
"Computerland", 6 City Arcade
Fore Street, Exeter
0392 77117

BITS AND BYTES
44 Fore Street, Ilfracombe,
N. Devon EX34 9JD. 0271 62801.

COMPUTER SYSTEMS (TORBAY)
Pump Street, Brixham
08045 6565/6.

CRYSTAL COMPUTERS
209 Union Street, Torquay
0803 22699.

SYNTAX LTD.
Midhurst, Grenofen,
Tavistock, Devon
0822 2392

DORSET
DENSHAM COMPUTERS
329 Ashley Road, Parkstone
Poole. 0202 737493.

ROBERTS ELECTRICAL (BLANDFORD) LTD.
14 Market Place, Blandford

ESSEX
AKHTER INSTRUMENTS
Unit 19, Arlinghyde Estates,
South Road, Harlow
0279 412639

CHELMSFORD COMPUTER CENTRE
3 Baddow Road, Chelmsford
0245 356834

COMPUTERAMA
88-90 London Road
Southend-on-Sea
Essex. 0702 335443

COMPUTERS FOR ALL
72 North Street, Romford
0708 752862

HANTS
MICRO VIDEO STUDIOS LTD.
60 Normandy Street, Alton, Hants
0420 82055.

HERTS
VIDEO CITY
45-47 Fishers Green Road,
Stevenage. 0438 53808

THE COMBINED TRADING CO.
10 & 11 Salisbury Square,
Old Hatfield. 07072 65551

KENT
ANIROG COMPUTERS
29 West Hill, Dartford.
0322 92513.

APHROS SOFTWARE CO.
47 Hawley Square, Margate
0843 294699

DATA STORE
6 Chatterton Rd., Bromley, Kent.
01-460 8991.

JUTEA LTD.
92 High Street, Bridge,
Canterbury, Kent. 0227 831183

LEAKES LTD.
63 Sidcup High Street,
Sidcup, Kent. 01-300 1142

LANCS
P.C.S.
39 Railway Road, Darwen
0254 776677

4MAT COMPUTING
67 Friargate, Preston
0772 561952.

SUMLOCK
Royal London House,
198 Deansgate,
Manchester M3 3NE
061 834 4233.

LEICS
DIMENSION
27-29 High Street, Leicester
0533 57479.

MOVIES COMPUTER CENTRE
5 Church Street, Melton Mowbray
0664 61169

LONDON
EROL COMPUTING
125 High Street, Walthamstow,
London E17. 01-520 7763

KAYDE HOME COMPUTERS
1 Station Approach, New Eltham,
London SE9. 01-859 7505

KELLY'S COMPUTERMARKET
227 Dartmouth Road
Sydenham, London SE26 4QY
01-699 4399/6202

MICRO ANSWERS LTD.
70-71 Wilton Road, Victoria
London SW1. 01-630 5995

MIDDLESEX
SCREENS MICROCOMPUTERS
6 Main Avenue, Moor Park
Northwood. 09274 20664

TWILLSTAR COMPUTERS
17 Regina Road, Southall
01-574 5271

N. IRELAND
D. V. MARTIN LTD.
13 Bridge Street, Belfast
BT1 1LT. 0232 226434

OXFORDSHIRE
SCIENCE STUDIO
7 Little Clarendon, Oxford
OX1 2HP. 0865 54022

SURREY
ANIROG COMPUTERS
8 High Street, Horley
02934 6083

COMPUTASOLVE
8 Central Parade, St. Marks Hill
Surbiton. 01-390 5135

COMPUTERAMA
17 Norkway, Banstead, Surrey
Burgh Heath 54717

SNOW-BEECH
1 East Grinstead Road,
Lingford, Surrey
0342 832476.

THE COMPUTER-WAY LTD.
73B North Street
Guildford, Surrey. 0483 62626

SUSSEX
WORTHING COMPUTERS
32 Liverpool Road, Worthing,
W. Sussex. 0903 210861

WALES
MICRO-CARE COMPUTING LTD.
18 Baneswell Rd., Newport,
Gwent NPT 4BP. 0633 50482.

TRYFAN COMPUTERS LTD.
57 Madoc Street,
Llandudno, Gwynedd
0492 70802

TYNE & WEAR
THE COMPUSHOP
10 Newgate Centre,
Newcastle-upon-Tyne NE1 5RE
0632 618673

WARWICKSHIRE
IMPULSE MICRO SYSTEMS LTD.
6 Central Chambers, Cooks Alley,
Wood Street, Stratford-Upon-Avon
0789 295819

W. MIDLANDS
CALISTO COMPUTERS
119 John Bright Street
Birmingham B1 1BE
021-632 6458

WORCESTERSHIRE
EVESHAM COMPUTER CENTRE
Crown Court Yard, Bridge St.,
Evesham. 0386 48635

YORKSHIRE
COM-TEC
6 Eastgate, Barnsley,
South Yorkshire. 0226 46972

EMPIRE ELECTRO CENTRES
783-789 Leeds Road, Bradford
BD3 8BY. 0274 662476

HARROGATE COMPUTER CENTRE
18 Cheltenham Parade,
Harrogate, N. Yorkshire
0423 57126

TOWERLIGHT LTD.
7 Crown Street, Hebden Bridge
W. York HX7 8EH
0422 843520

COMPUTERS FOR ALL DEALERS